DEVELOPMENT OF AN IMPROVED SEQUENCING BASED HEURISTIC AND COMPARISON OF ITS PERFORMANCE WITH OTHER SEQUENCING AND LOT SIZING ORIENTED HEURISTICS IN MRP CONTEXT

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

Master of Technology

by

Sharat Sinha

to the

DEPARTMENT OF INDUSTRIAL & MANAGEMENT ENGINEERING

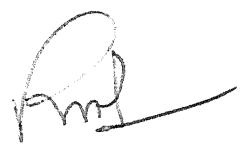
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

MAY, 1994.

IME-1994-M-SIN-DEV

CERTIFICATE

It is certified that the work contained in the thesis entitled " Development of an improved sequencing based heuristic and comparison of its performance with other sequencing and lot sizing oriented heuristics in MRP context ", has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.



(Dr R.R.K. Sharma)

Industrial & Management Engineering

Indian Institute of Technology

Kanpur .

April, 1994

ABSTRACT

Material requirements planning is extensively used in the industry as the core part of the total manufacturing planning and control. To obtain maximum advantage from it, it is important that its working be efficient. Many Lot Sizing and Scheduling rules, their combinations and their modified forms have been tried to give better results in a MRP environment.

The present work is a simulation study of a MRP system to study the effect of coordination between various components reaching a assembly centre. In it, it has been attempted to reduce the time lag between components so that time of waiting at a centre by the components can be saved and by doing it the performance of the MRP system can be improved. In order to it, a coordination index along with a standard scheduling rule is used to decide the priority of components at levels earlier than the level at which it gets assembled.

On application of this approach it was observed that it gave significantly better results on important parameters like total units lateness and total aggregate lateness. It even improved the average capacity utilization of machines. For comparison, two other approaches of which one tries to selectively reduce lot sizes and the other tries to expedite the late reaching components, were considered.

It is hoped that the developed approach would be of some practical help to a practitioner of MRP and would provide a new direction for future research work in this area.

ACKNOWLEDGMENTS

First and foremost I would like to thank my guide Dr R.R.K Sharma who provided constant reassurance and support, especially when the going was not smooth. Words are not sufficient to express my gratitude towards him , without whose help this work would not have been possible.

I would like to take this opportunity to thank all the faculty members, who have made my stay at I.I.T Kanpur an informative and memorable one. I also express my deep gratitude towards the whole I.M.E family for being very helpful and cooperative and making my stay at I.I.T a very pleasurable one.

Sharat Sinha

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In production context, inventory is an idle resource. It includes the production material like tools, purchased parts, raw materials, product in process. The fact that the resource is idle does not mean it is serving no purpose. It serves as an insurance policy against unexpected breakdowns, delays and other disturbances that could disrupt the production process. Since cost is associated with the disruption of the production process and also with maintaining a stock of idle resources, the task of operations management is to secure a economic balance between the cost of production time loss and the cost of preventing it.

The most important function of inventory is insulation. A reserve of material can be tapped whenever delay in a preceding stage threatens to curtail operations in the following stage. Material buffers are used to cushion the production process from uncertainties of material deliveries, to decouple progressive stages of product development from disruption in previous stages, and to provide a steady supply of finished output for the unsteady demand of customers.

Operations management, since it involves operational, material and financial aspects of a enterprise, is the heart of the manufacturing organization. An efficient operations management is essential for the health of an enterprise.

Large manufacturing organizations, such as one engaged in aircraft manufacturing, have series of machining and assembly operations. Scheduling and inventory control become important aspects of operations management in these manufacturing

aspects of operations management in these manufacturing organizations.

In an MRP system inventory control and scheduling techniques are integrated. It consists of a series of steps which start by determining what finished products are needed to meet the demand of different time periods, and arrives at a time schedule of the finished product components needed at each assembly for each time period.

The material requirements planning system was developed to improve the performance of a production system in a complex manufacturing setup involving both assembly and manufacturing operations. Distinctive characteristics of these complexities make the traditional methods of inventory control inadequate.These are, (i) the demands and resulting schedules for all components are directly dependent on the schedule of requirements for higher level assemblies. (ii) Due to limited machine capacities, queues are formed before them. (iii) Sequencing decisions affect inventory related performance of the shops. The occurrence of dependent demands is used by MRP system for coordinating the inventory ordering policies for all the components of the manufacturing process. Coordination is essential in the case of dependent demands as shortage of a single item can stop the assembly process, which leads to low capacity utilization and increase in inventory carrying costs. In MRP environment, the demand for the end products as well as intermediate products/assemblies is dynamic.

1.1 MRP Operation

A bill of materials describes how a product is made from its component parts and assemblies. An MRP system links the planned production schedule with the bill of materials needed to make the product and examines the manufacturing inventory to see which parts and raw materials have to be ordered . By considering when various components of the end product are scheduled to be produced and the necessary lead times for supply, it time phases replenishment orders, so that parts and materials are available when they are needed at work stations. Ideally, replenishments would arrive at the stations exactly when they are required for use, thus minimizing the amount of work in process inventory. Since ideals are rarely attainable, the system reexamines the production schedule, continuously or periodically, to recognize schedule disruptions and irregularities as soon as possible; this reduces stockpiles of inventory in the production area. Success of adjustments relies on the accuracy of the forecast, bill of materials, processing times, routines and inventory records.

In the total manufacturing control system the position of MRP is right in the middle of the total control system. Manufacturing control results from forecasts that guide capacity planning and inventory control. Master scheduling considers available capacity , stock status, forecast demand and customer orders to develop a production schedule. The MRP system issues planned replenishment orders based on information about job status and work in process inventory levels. Production results from releasing the planned orders in terms of work loads. Data needed

to control operations are obtained by measuring work centre output, work in process inventory, and job status.

The MRP system envelops the entire manufacturing organization where it receives certain inputs and after processing them gives certain outputs for implementation. The quality of these outputs determines the effectiveness of a MRP system. For MRP to generate proper output, accurate and timely availability of the inputs is essential. Accuracy of these inputs greatly affects the performance of MRP. Below we discuss important decision making areas in a MRP system. These are lot sizing, sequencing, determination of planned lead times and the capacity planning.

1.2 The Lot Sizing decision

In the lot sizing procedure the timing and size of the replenishment order is decided given the discrete period demands i.e requirements are converted into a series of replenishment orders. For one component this problem involves determination of the grouping of the requirements data into a schedule of replenishment orders that minimizes the sum of setup costs and inventory carrying costs. In a multistage assembly environment the decision regarding lot sizing becomes all the more important as lots of the previous machines affect the lot sizing decisions of subsequent operations. Thus obtaining optimal lot size solutions becomes more difficult.

Various lot sizing situations arise and optimal lot sizing decision is determined for each case by a separate procedure. It was argued above that lot sizing rules for a

production system where "n" machines are arranged in series is likely to be more complicated than the lot sizing rules for single machine case. Even for single machine case situation is different when demand is deterministic and when it is probabilistic. The case of deterministic demand is further classified into two cases such as (i) uniform and (ii) dynamic. Simple EOQ formula gives optimal lot size decisions for the case of uniform demand. This rule produces "robust" lot sizes and hence can be used even there is little uncertainty about the demand figures. The dynamic programming based on Wagner Whitin rule [13] is popularly used for the case of dynamic demand. When the demand is uncertain, the lot sizing rules consider the additional factor of service level. Hadley and Whitin [7] give a good classification of these rules.

Since the situation in MRP is a case of multiple machines in series and with many products and demand may be assumed to be dynamic (which itself is a simplifying assumption), the lot sizing decision is not an easy one, atleast computationally. Hence simple lot sizing rules such as EOQ and Wagner Whitin rules are repeatedly applied even though it is known that they will produce sub optimal solutions. Many lot sizing rules that are used in MRP contexts are lot for lot, Silver Meal heuristic, Wagner Whitin algorithm, Part Period balancing etc.

1.3 Scheduling decision

In a multistage problem where there are a large number of processing machines, since processing a lot through a work centre takes time, the processing of other parts which use the same work centre is delayed and they have to wait in a queue. This

delays the availability of assemblies at upper levels and leads to increased inventory carrying costs.

Thus the sequencing decision affects the due date performance as well as the inventory carrying costs. Various sequencing rules used in MRP context are - SPT (shortest processing time first), BDD (earliest job due date first), CRR (critical ratio rule), ORIB (number of operations remaining in a branch) etc.

1.4 Planned lead time

Planned lead times are used by the MRP system for the purpose of planning. These lead times are used to determine order release dates. The lead time of a manufactured item is made up of a number of elements. These are - waiting time in queue, running (machining, fabrication, assembly etc) time, setup time, waiting time (for transportation, inspection etc).

1.5 Capacity Planning

Capacity requirements planning is the function of determining what capacities will be required, by work centre by period, in the short to medium range, to meet current production goals. The output of MRP system indicates what component items will have to be produced and when, and this output can therefore be converted into the capacities to produce these items (Orlicky [8]).

Some of the performance criteria over which these decisions are judged are number of late orders, total aggregate lateness, total units lateness, total inventory carrying cost,

number of stockouts etc. The performance over these parameters is affected by the choice of lot sizing rules, scheduling rules, planned lead times and capacity planning decisions. It has been shown that lot sizing rules and scheduling rules interact with each other and that some lot sizing rules go well with certain types of sequencing rules (Biggs[2]). Thus the decision maker is actually confronted with the decision regarding the combination rather than the individual decision of lot sizing or scheduling. We discuss these issues in detail in chapter 2 on the literature survey.

1.6 Scope of Present Work

Most of the earlier works in MRP have been to determine the effects of various rules in different decision categories and seek to choose the best combination of several rules. Very few works are available that help to improve a given solution. The work of Sharma and Potty [11] was an exception where they have appropriately modified the inventory carrying costs of the products used for lot sizing decisions based on the results of scheduling decisions. This had resulted in significant improvement of results over the best known combination of decision rule for lot sizing (which is Wagner Whitin [13]) and sequencing (which is the Critical Ratio rule, see Goodwin and Weeks [6]). Later, Yoganandan [14] has modified the priorities in the sequencing module of MRP based on their earliness/lateness at an assembly centre and obtained superior results than the rules given in Sharma and Potty [11].

We refine the idea given in Yoganandan [14] to develop

the concept of coordination index and develop rules which give results that are comparable to the ones given by Yoganandan, but give significantly better results on the dimension of total units lateness.

In the present work the new approach of achieving coordination amongst the child components attempted to reduce the range (in terms of time) of child components reaching a assembly centre in a multistage assembly environment. For lot sizing and scheduling, standard rules which had shown good results in earlier work done were chosen (the details are included in the chapter on literature survey). By using the coordination range for calculating a coordination index which was later used as a priority value for lots, a further improvement in the performance of the system was achieved which establishes a relation between coordination and performance of the system. Two heuristics using this approach were developed and one based on considerations of lead time was developed.

1.7 Reasons for using the iterative heuristic approach

Mathematical model which takes into account the lot sizing of parts at different levels of the product structure and sequencing rule for scheduling the parts on each work centre is quite complex and the solution procedure quite expensive.The heuristic approach based simulation is computationally much more attractive.

1.8 Organization of the Thesis

In chapter 2 we briefly give the literature survey and in chapter 3 we give the prerequisites, tools and terminology of MRP.

In chapter 4 we describe the new heuristics developed by us. In chapter 5 we give the essence of the MRP simulator whose program listing is given in Appendix A. Basic skeleton of this program was proposed by Dr R.R.K.Sharma in which we have embedded the various heuristics. In chapter 6 we analyze the results obtained and discuss these results. In chapter 7 we present our conclusions and discuss the implication for future research. The details of twenty test problems with results and "t" test results are given in Appendix B.

CHAPTER 2

LITERATURE SURVEY

The impact of product structure on the performance of priority dispatching rules was studied by Russell and Taylor [10]. Two type of BOMs were studied. It was found that as BOM gets taller with more levels, the probability that the job will finish after its due date increases. It was also reported that performance of shortest processing time (SPT) sequencing rule improves as product structure gets taller.

The effects of product structure and sequencing rule on assembly shop performance was studied by Fry, Oliff, Minor and Keong [4]. In this paper an effort was made to study the effect of product structure on the performance of fourteen selected priority rules in a simulated multistage job shop (MRP environment). The product structures chosen can be classified into three groups - flat, tall and complex. The results established a relation between product structure and scheduling rules in the performance of MRP system and specific rules are indeed better suited to certain product structures.

The results of the study indicated that there is a strong relationship between product structure and sequencing rule performance. Taller structured BOMs tended to be more tardy than flat BOMs using the same due date allowance, a finding supported by Russell and Taylor [10]. But contrary to to Russell and Taylor this study indicated that the performance of SPT does not improve

as product gets taller. SPT never performed well for flat BOMs for any performance measure. Despite being one of the best rules for the single stage job shop, results from this study suggest that that SPT is not appropriate for a shop that performs assembly operations.

This study indicates that branch critical ratio rule (BCR) performs well on the criterion of mean absolute lateness (MAL).

Dispatching in a multistage job shop where machine capacities are unbalanced was studied by Fry, Philipoom and Markland [5]. In this paper, the performance of selected dispatching rules in an unbalanced multistage job shop was evaluated to determine whether the performance was consistent for jobs which were routed through the bottleneck and for jobs which bypassed the bottleneck. The results indicated that sequencing rules which are based on due date tend to be the overall better performers for both type of jobs. No rule was found to be superior in all cases for all measures. Another result was that the results for the jobs which pass through the bottleneck were not the same for the jobs which bypass the bottleneck. Also the degree of capacity imbalance or due date allowance did not cause a significant change in the ranking of the sequencing rule performance.

R.R.K.Sharma and V.S.Potty [11] developed multipass heuristics for improved stockout performance. In this work two heuristics were used to improve the stockout performance. In them

lot sizes were selectively reduced in steps to get the desired improvement. The criterion for choosing the part for revising the lot size was based on the difference in its actual and theoretical inventory carrying cost. After choosing the part its holding cost was changed to revise the lot size. The results indicated that increase in the holding cost reduces the lot size which improves the stockout performance but excessive decrease in lot sizes may increase the number of setups, and adversely affect the due date performance (as more capacity may be lost in setups now). To take care of this the modification in lot size was limited to two times. The study also indicated that critical ratio rule (CRR) consistently performed better on a variety of performance criteria.

The objective of this chapter is to describe the basic concepts of MRP, terms used in MRP and the methodology of MRP. The objective is to provide a overview. For finer details one can refer to orlicky[8].

3.1 Prerequisites

i) Existence of a Master Production Schedule based on forecast of product demand is essential for MRP. It states how many items are required and when they are required to be produced or purchased.

ii) Inventory record for all components with inventory status data is required as an input for MRP.

iii) Each item is required to be unambiguously identified by a unique component number. A BOM indicating all the components and the structure giving all the details regarding processes involved, child assemblies, processing and routing sequence is required as a input for MRP.

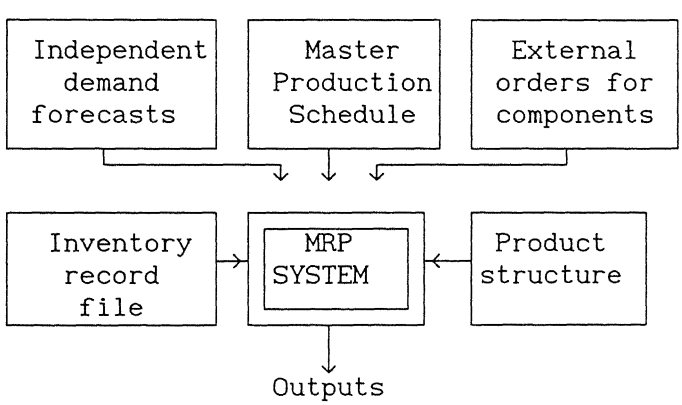


Figure 3.1

MRP Inputs

3.2 Terminology for MRP

3.2.1 Pegging

For many items the demand is created from many sources. The source identity gets lost when the total requirements are collected for an item's material requirements plan. For some occasions it may be important to know the status of the status of the items intended for certain destinations. For instance, if production delays are likely to create a shortage of a particular item, it would be helpful to know which assemblies, finished products and, ultimately, customer orders may be adversely affected. This is accomplished by pegging where an item is 'pegged' with information to identify its parent in a given time bucket, see Orlicky[8].

3.2.2 Netting

Netting is the process in which net requirements are calculated by subtracting quantity available from the gross requirements. Gross requirement figure is based on the demand forecast for the item.

3.2.3 Offsetting

The determination of the appropriate time for release of planned orders with the objective of their timely completion to meet their demand is called offsetting. The planned order release determined by subtracting the lead time from the data of earliest

net requirement which it is expected to satisfy, see Orlicky [8].

3.2.4 Lot Sizing

Lot sizing is based on the principle of economy of scale. In it by giving due consideration to ordering cost, (setup cost in case of in house production) holding and other costs involved lot sizes are produced in order to minimize the overall cost involved in the production of an item. There are many methods of obtaining economic lot sizes. A list of nine such methods is given in orlicky [8]. Here Wagner Whitin [13] is discussed which gives optimal solution for the single machine dynamic demand case. This is used in the simulation model in the present work.

3.2.4.1 Wagner Whitin algorithm

This algorithm uses an optimizing procedure based on dynamic programming. It evaluates all possible ways of ordering to cover net requirements in each period of the planning horizon. Its objective is to arrive at the optimum ordering strategy for the entire net requirements schedule. The Wagner Whitin algorithm minimizes the total cost of setup and carrying inventory, and is used as a standard for measuring the relative effectiveness of the other discrete lot sizing techniques, see Orlicky [8].

3.2.5 Scheduling

Scheduling in a MRP system involves decision regarding priority for processing of components i.e it decides in which order components will be processed once they reach the work centre. In the simulation model for the present work scheduling

has been used for decisions regarding queue formation in front of a machine. There are many scheduling rules for job shop environment, a detailed list of which is available in Baker [1]. In the MRP environment the Critical Ratio Rule (CRR) has been shown to do well on many performance criteria, see Biggs [2].

3.2.5.1 Critical Ratio Rule

Critical Ratio is given by,

$$\text{Critical Ratio} = \frac{\text{Lead time remaining - Work remaining}}{\text{Work remaining}}$$

The job having the least critical ratio is given the higher priority. This is based on the principle of processing that component first which is needed the most in the next period. This due date based rule is shown to perform well with respect to all the performance criteria (Biggs [2] and Goodwin and Weeks [6]).

3.2.6 Explosion

Explosion is the process of translating product requirements into component part requirements, taking existing inventories and scheduled receipts into account.

In the explosion process, the demand data from master production schedule is taken as the product requirement. Using this data and data from BOM file, component requirements are calculated.

In the present work three heuristics have been developed. Two of the heuristics (which were developed) were based on considerations of coordination range and one on lead time. We have also tested the performance of the heuristics developed by Potty [9] and Yoganandan [14].

In the work that was done by Potty, in the heuristic that he developed, it was attempted to improve the stockout performance of the system by selectively reducing the lot sizes. The criteria for selecting the components for lot size modification was based on the difference in actual and theoretical inventory carrying costs. The components for which the difference exceeded ten percent (this limit is set arbitrarily) were chosen for lot size modification. The lot sizes were modified by increasing the holding costs of the relevant components. This approach was reported to have given better stockout performance over the original result (the one without the lot size modification).

In the work which was done by Yoganandan [14], the scheduling rule used was modified for better performance. In this approach after the scheduling was done, the assemblies arriving later/earlier than fifty percent (again this figure of 50% is arbitrary) of the subassemblies arriving at an assembly centre were identified, and for the subsequent rescheduling their criticallity at the previous processing centres was increased/decreased. The rescheduling was attempted for a maximum of twenty times.

of twenty times.

In the present work the information from Potty [9] and Fry et.al [4] was used for choosing the standard lot sizing and scheduling rules. It was reported there that due date based scheduling heuristics are overall better performers. The first simulation run was done using these standard rules. In the first run the coordination range (a term used for difference in time between the latest and earliest lot reaching a assembly centre) for various assembly centres was calculated. The mean of the times of lots reaching a particular assembly centre was calculated for each assembly centre. The priority of the lots at previous processing machines was decided by the latest reaching lot of a particular assembly. Depending on the time by which the latest reaching lot of a component varied from the mean value for a particular machine and whether it was more or less than the mean value, its priority was modified at the previous processing machines. In the finally developed approach only lots of those components were modified whose latest reaching lots had reached at a time greater than the mean value.

Seven parameters were observed as indicators of system performance. These were :

i) Total aggregate lateness - This is the sum of all the times by which the lots get delayed. The lower is the value of this parameter, the better is system performance.

ii) Total setup cost - It is sum of all the costs involved in setting up the machines for various operations.

iii) Total units lateness - This quantity is the sum of the

product of units and the time period units by which the components get delayed.

iv) Total inventory carrying cost - This is sum of all the inventory carrying costs.

v) Theoretical inventory carrying cost - The theoretical inventory carrying cost is given by the lot sizing model which seeks to balance the inventory carrying cost and setup cost while ignoring the sequencing aspects of the MRP system.

vi) Average utilization - For the general machines capacity utilization is given by the percentage of the total available time for processing when the machine was busy. Average utilization is the average of all the machine utilizations.

vii) Average coordination range - Coordination range for a particular assembly centre is the difference between the time of latest reaching lot at the assembly centre and the earliest reaching lot at the assembly centre. The average coordination range is average of the coordination ranges of all the assembly machines used.

In the present work, it was attempted to reduce the coordination range and see its effect over all the remaining parameters which have been described earlier.

The five heuristics which were tested are as follows.

4.1 Potty's Heuristic (H1) [9]

Let R be the set of parts with difference in actual and theoretical holding cost exceeding 10 % and S be the set of parts already considered twice in the iterative procedure. Let K[j] denote the number of times the part j has been considered in

the iterative procedure, and BestSol denote the best solution arrived at with respect to total units lateness. The steps of the heuristic are described below.

Step 1: Initialization. Set K[j] to zero for all j. Using Wagner Whitin for lot sizing and CRR for scheduling simulate the material requirements planning system to prepare the set R and initialize the following.

S ← 0, K[j] ← 0 for all j, BestSol ← infinity

Step 2: Select part j from set R which has the highest percentage difference in theoretical and actual holding costs. Increase the holding cost of part j in steps of 0.2 until the lot size changes. Set K[j] ← K[j] + 1 and if (K[j]=2) then updateS ← S + {j}. If the current solution is better than the BestSol w.r.t to total units lateness then update BestSol ← Current solution.

Step 3: Simulate and prepare the set R afresh and let R ← R - S

Step 4: If R = 0 then stop else go to step 2.

4.2 Yoganandan's heuristic (H2) [14]

After the scheduling is done, the assemblies arriving later/earlier than the fifty percent of the subassemblies arriving at an assembly centre are identified, and for the subsequent rescheduling their criticallity at the previous processing centre is increased or decreased. The rescheduling is attempted for a maximum of twenty times.

4.3 Heuristic developed (H3)

4.3.1 The Coordination Index approach

In this approach, based on the coordination range

for a assembly centre and the latest reaching lot of a component, coordination index for the lots of the component is calculated. This value along with the scheduling rule (CRR in the present case) is used for deciding the priority of the lots in the subsequent simulation runs.

Let A be the set of components which undergo processing. Let C be the set of coordination indexes corresponding to each member of A. Let BestSol be the best solution arrived at with total units lateness.

Step 1: Initialization. Set C[j] to one for all j, where j denotes a component no. Using Wagner Whitin algorithm for lot sizing and CRR for scheduling (i.e for deciding priority) simulate a material requirements planning system. Initialize BestSol ← infinity.

Step 2: First run - In the first simulation run use the scheduling rule with coordination index value as one for deciding the priority of lots when they join queues in front of general machines.

Step 3: Next run - In subsequent simulation runs set coordination index to

C[j] ← 1 + (latest reaching lot of j - mean time of reaching corresponding to assembly machine at next level)

Set priority value to

Priority value of j ← (Product of Critical Ratio and coordination Index) * (-1)

where, Critical Ratio is given by,

$$\text{Critical Ratio} = \frac{\text{Lead time remaining - Work remaining}}{\text{Work remaining}}$$

Step 4: Perform step 3 twenty times.

In the heuristic H3 we kept the lead times fixed and did not vary it across simulation runs.

4.4 Heuristic based on lead time (H4)

In this, heuristic along with applying coordination index for deciding priority as in H3, planned lead times for the lots was revised depending upon the waiting time of the lot in the queue in front of the general processing machines. The planned lead time increase was proportional to the waiting time in the queues.

4.5 Heuristic based on modifying the priority according to coordination index of every lot (H5)

In this heuristic, coordination index corresponding to each lot was calculated (unlike H3 where coordination index corresponding to each component was calculated). Rest of the method was same as H3.

CHAPTER 5

IMPLEMENTATION

In the present work, in order to test the performance of the developed heuristic, a simulated job shop with assembly and general processing machines was used. A simulator of such a shop was already developed by Dr R.R.K.Sharma in PASCAL. This program was debugged and linked with the heuristics developed in this work.

5.1 The Simulated Job Shop

The simulated job shop had assembly machines and general processing machines whose number could be changed depending upon the requirement of the problem. A problem consisted of all the necessary inputs for a MRP system i.e MPS, BOM (product structure), processing and setup times for various machines and the cost data which included the holding costs, order costs, setup costs, processing costs, cost of component etc.

The BOMs (product structures) chosen were of different types as different type of product structures have been found to behave differently with a scheduling rule as reported by Fry et.al [4]. While designing the problems the time and cost data as well as the demand figures for various components was randomly generated.

This data of the the problem was given as a input to the system.

The simulation carried out was a event based simulation. The sequence of events that took place in the simulation are described in the paragraphs to follow.

simulation are described in the paragraphs to follow.

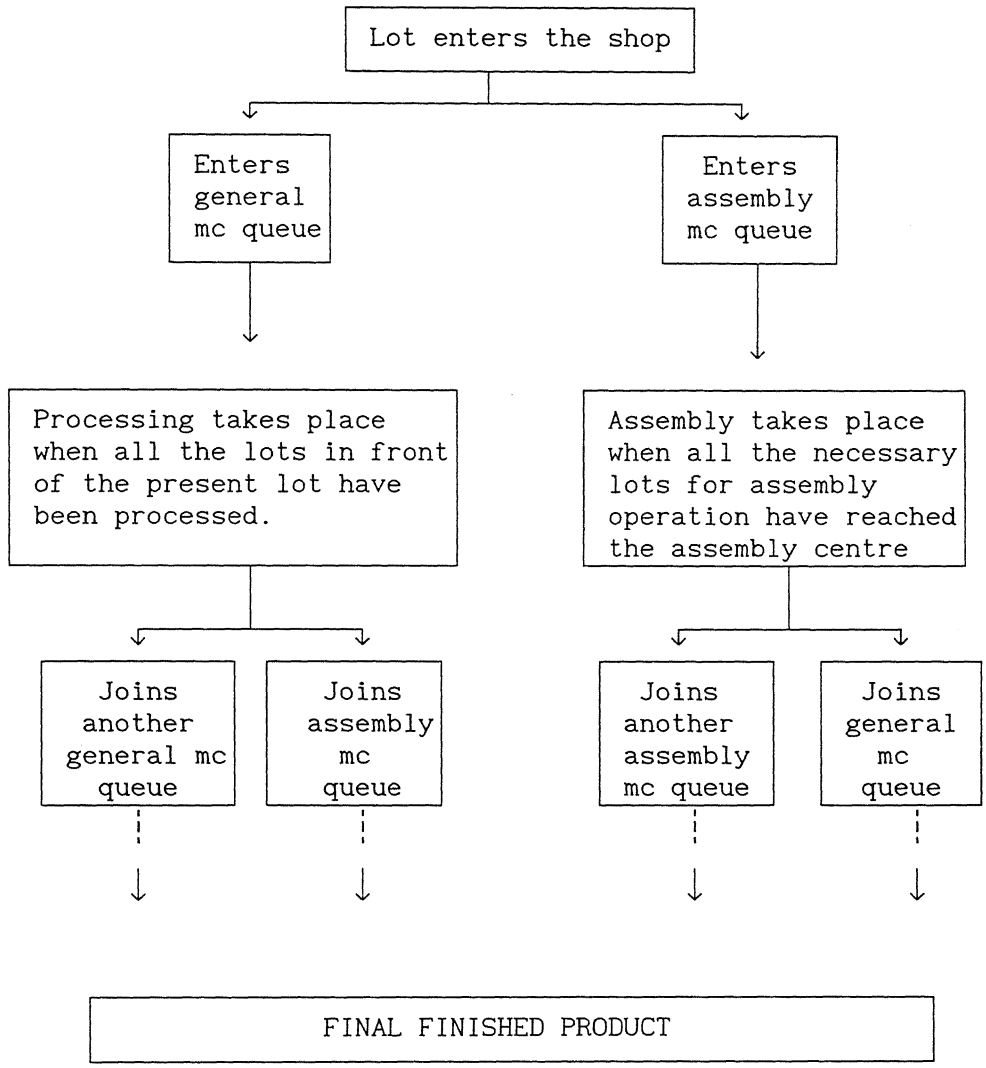
The first operation was to read all the data from the relevant files. This was followed by the explosion process where the product requirements from the MPS was translated into component level requirements.

Once explosion was over, based on the data from MPS and data of costs, lot sizing was done by using Wagner Whitin algorithm. This resulted in economic lot sizes to be produced in different periods.

Now based on the data related to product structure, those components were chosen which had no child assemblies. Depending on which general machine was required for its processing first it was put in the queue in front of that assembly machine. The position of the lot in the queue was decided by the priority value based on Critical Ratio Rule in the first run and by combined effect of critical ratio and coordination index in the subsequent runs. After all the lots in front of the present lot were processed, the present lot was processed. This was followed by a trip to a assembly or general machine depending upon the requirement. This process continued till the final finished was produced. The general approach for lot routing is as shown by the diagram on next page.

Figure 5.1

Simulation Scheme



5.2 Test problem development procedure

The heuristics tested are based on the lot sizing decision and the sequencing decision in the MRP systems. Although heuristic H4 tries to incorporate the effects of lead time decision also, the capacity decision is totally ignored in the developed/considered heuristic.

However it was found that the developed heuristics were totally ineffective when machine capacity utilization variance was high. Hence we attempted to control this factor in the test problems developed, and tried to generate test problems that kept utilizations across machines at uniform level. This required lot of time in developing the test problems. Typically it took four days to develop a test problem.

CHAPTER 6

RESULTS AND DISCUSSION

On the simulation model developed, twenty test problems were tried. While designing the problems it was tried to keep a variety of product structures, as it has been reported by Fry el.al [4] that different product structures behave differently with different scheduling rules. By doing this it was tried to see if the developed heuristics were consistent. The various time figures (processing times, setup times etc) and cost figures (holding costs, processing costs, setup costs etc) were randomly generated. The details of test problems and the results obtained (including t test results) appear in Appendix B.

5.1 Comparison of Results

For comparing the results of the three heuristics, test of significance (t test) was performed over the data of percentage improvement given by each heuristic over the combination of Wagner Whitin rule and CRR rule whose results appear under the head "original". The three heuristics for which data have been considered are H1, H2, H3. Since H4 and H5 did not give any improvement over H3 or H2, their data have not been included due to lack of time.

The twenty problems which were tried have been included in Appendix B. The analysis of the results of the three heuristics on seven parameters is as follows.

5.2 Discussion

On the basis of results generated by performing the t - tests on percentage improvement figures on various performance

criteria, following can be inferred.

i) Total units lateness - H2 though gives better results than H1, the results are not significant at 5% significance level. Results given by H3 are significantly better than both H1 as well as H2 over this parameter at 0.5% significance level. In this case, H1 though gives better results than original in nine out of twenty cases, and inferior results only in six cases, the mean improvement is negative. This lends credence to the fact that the rules are suitable for certain categories of product structures as has been found by Fry et.al [4].

ii) Total aggregate lateness - The results obtained by H2 are though better than H1, they are not significantly better at 5% significance level. H3 gives significantly better performance than both H1 as well as H2 at 0.5% significance level.

iii) Total inventory carrying cost - Both H2 as well as H3 give better results than H1 at 0.5% significance level. H3 though gives better results than H2, its results are not significantly different at 5% significance level.

iv) Average utilization - H2 gives better results than H1 but results are not significantly different at 5% significance level. H3 gave significantly better performance than both H1 as well as H2 at 0.05% significance level.

v) Average coordination range - H1 gave significantly better performance than H2 at 0.05% significance level. H3 gave better performance than both H1 as well H2 at 0.05% significance level.

vi) Theoretical inventory carrying cost - Except for H1 where due

to lot size change this parameters change, other two do not show any modification in this parameter. Therefore, for this parameter t test was not applied.

vii) Setup cost - For this parameter also, except for H1, other two heuristics do not show any change. Therefore, this was not considered for t test.

In all the problem instances we had considered, it has been uniformly observed that if the machine utilization variance was high then it resulted in all the heuristics H1, H2 and H3 being ineffective in giving improved results. Hence, machine capacity utilization had a substantial effect on the efficacy of these heuristics. This has also been the observation of Potty [9] and Yoganandan [14] who had prepared problem instances where all machines had relatively uniform capacity utilizations. Similarly we had included sufficient machine capacities to keep machine utilizations at a uniform level, the heuristics produced the results which have been documented here. In future work one has to consider the effect of lopsided machine utilizations and derive heuristics that give results in such conditions.

Potty [9] and Yoganandan [14] had kept the lead times constant across the lead times constant across all the components and assemblies. We had kept it variable and in proportion to waiting times before the machines for each component and subassembly. This could be an influencing factor that possibly has lead to improved performance over the performances obtained in Potty [9] and Yoganandan [14]. Future work could be done to refine the lead time heuristics even further.

Due to lack of time we could not combine the heuristics H1 and H3 into one procedure where we sequentially apply H1 and H3 till improvements are possible. When lot size modifications cease to improve results, we could initiate sequencing oriented changes to seek improvements and so on.

Yoganandan [14] had shown that sequencing modification based heuristic H2 (as reported here) does not improve the performance when faced with flat structures. We have identified 11 out of 20 problems attempted here where lot sizing modification based heuristic H2 (as reported here) does not improve the performance. There could be several reasons for this.

i) The selection of parts qualifying for holding cost modification (parts having 10% difference in actual and theoretical holding costs) tends to be a bit arbitrary.

ii) The product structures of each problem instance could be different.

One may conduct simulations controlling these factors to determine the strength of their effects. One of the findings is that for heuristic H1 the increase in costs was associated with reduced coordination index for eight out of twenty problem instances. It indicates that coordination index is a useful measure of shop performance only if assemblies are "in time". If all the assemblies are coordinated "late", then it ceases to be a useful measure.

In this research we have presented a refined heuristic that seeks to improve the overall shop performance. Since our sample size is limited its generalizability is tentative even though its performance is impressive.

Varying lead times of each lot (heuristic H4) and changing priority of each lot (heuristic H5) does not give significantly better results, although we did not document these results due to lack of time.

There is scope to do research to find diagnostic measures which tell us which heuristic to use (H1, H2 or H3 or any other). In this work we have attempted to determine the efficacy of two such measures, i.e, (i) Difference in actual and theoretical inventory carrying costs and (ii) Coordination range defined as the variance/range of arrival times of different subassemblies at an assembly centre, with substantial success.

It has been recognized in literature that certain heuristics are more suited to certain product structure and that all the decisions in MRP systems (lot sizing, sequencing, lead times and planning machine capacities) interact with each other. A detailed investigation could throw more light on these issues.

Significantly superior results obtained by heuristic H3 shows that computationally attractive heuristics have promise in giving improved performance.

CHAPTER 7

CONCLUSIONS

One of the primary aims of this research was to develop a approach by which coordination among the components can be achieved, which can improve the system performance. It was observed that by reducing the variance in arrival times of components reaching a assembly centre (i.e by ensuring better coordination), system performance can be improved. The approach ultimately followed was of reducing the time range at a assembly centre (referred to as coordination range in the present work) which was found to give better results than the two works earlier done.

Out of seven parameters considered, total units lateness, total aggregate lateness and average utilizations were the parameters for which coordination approach gave significantly better results. The coordination approach was also found to have improved average capacity utilization.

Scope for future work

Some related areas where some future work can be done are as follows :-

i) Use of modification in Lot Sizing to reduce coordination range for better system performance.

ii) Since scheduling rules are sensitive to product structures, effect of product structures on coordination range can be studied.

iii) It was found that by having a better coordination even the average capacity utilization gets improved significantly. Study of various factors which can improve the capacity utilization can also be under taken.

also be under taken.

iv) In many cases the variability in improvement is high, which implies that these rules do not perform well on all product structures. Hence, one needs to develop a framework which suggests a matching rule to a given product structure. This research has gathered sufficient evidence to show that such a framework is possible.

REFERENCES

1. Baker. K. R, "Introduction to Sequencing and Scheduling" , John Wiley and Sons, 1974.

2. Biggs. J.R, "Priority rules for shop floor control in a material requirements planning system under various levels of capacity", International Journal of Production Research, 1985, 23(1), 33-36.

3. Bowker and Lieberman, "Engineering Statistics", Prentice Hall, England.

4. Fry. Timothy.D, Oliff. Michael.D, Minor. Elliot.D, Leong.G.Keong, "The effects of product structure and sequencing rule on assembly shop performance", International Journal of Production Research, 1989, VOL 27, 671-686.

5. Fry. Timothy.D, Philipoom. Patrick. R, Markland. Robert.E, "Dispatching in a multistage job shop where machine capacities are unbalanced", International Journal of Production Research, 1988, VOL 26, No. 7, 1193-1223.

6. Goodwin. J.S and Weeks. J.K, "Evaluating scheduling policies in a multilevel assembly system", International Journal of Production Research, 1986, 24, 247-257.

7. Hadley and Whitin, "Analysis of Inventory Systems", Prentice Hall, India, 1980.

8. Orlicky. Joseph, "Material Requirements Planning" , Mcgraw Hill Book Company, 1975.

9. Potty. V.S, "Simulation of heuristics, lot sizing and sequencing procedures in MRP", M.Tech dissertation, I.I.T, Kanpur, 1990.

10. Russell.R.S and Taylor.B.W, "An evaluation of sequencing rules for an assembly shop", Decision Sciences, Vol 16, 196-212, 1985.

11. Sharma. R.R.K, Potty. V.S, "Multipass Heuristic for improved stockout performance", In Dr V.Raju (Ed):Design automation and computer integrated manufacturing, Tata McGraw Hill Publishing Company, New Delhi, 1991, 275-282.

12. Vollman. Thomas.E, "Production Planning and Control Systems", Galgotia Publications Pvt Ltd, 1989.

13. Wagner. H.M and Whitin. T.M, "Dynamic Version of Economic Lot Size Model", Management Science, October, 1958, 89-96.

14. Yoganandan.S.R, "Coordinated Scheduling in MRP Systems", M.Tech dissertation, I.I.T, Kanpur, 1992.

APPENDIX A

SIMULATOR DEVLOPED

The following is the program in PASCAL which was used for simulation.

```
program mrp(input,output);

const
        max_mc_no                              = 15;
        max_no_of_proces_on_any_assly          = 5;
        max_assly_no                           = 22;
        max_subassly_no_into_an_assly          = max_assly_no;
        max_period_no                          = 50;
        infinity                               = 40000000;
        max_statistics_pt_no                   = 4;
        computing_points_per_period_in_w_w     = 100;
        max_no_of_lots                         = max_period_no;
        max_queue_length                       = 125;

        inventory_cost_fraction = 0.12;
        no_of_hours_in_a_year = 8640;
        no_of_hours_in_a_period = 24;
        collect_statistics_from_time = 10.0 * no_of_hours_in_a_period;
        lead_time=50.0;
        simulation_time=infinity;
type
        range_of_assly_mcs = 16..20;

        ptr_to_event = ^event_record;

        event_type = (collect_statistics_from_now_on,
                      lot_enters_the_shop,
                      lot_enters_the_general_mc_queue,
                      lot_enters_the_assly_mc_queue,
                      processing_on_general_mc_starts,
                      processing_on_assly_mc_starts,
                      processing_on_general_mc_gets_over,
                      processing_on_assly_mc_gets_over,
                      lot_enters_the_finished_product_list);

        process_data = record
                         mc_no : integer;

                         setup_time,
                         processing_time_per_unit,
                         setup_cost : real;
                       end;

        child_assly = record
                        assly_no,
                        qty_to_be_contributed : integer;
                      end;

        subassly_data = record
                          assly_no,
                          qty_av : integer;
```

```
                    assly_no,
                    qty_av : integer;
                  end;

lot  = record
         assly_no,
         original_size,
         qty_left,
         no_of_agm_visited,
         current_mc_no   : integer;

         processing_complete : boolean;

         aggregate_lateness_of_lot,
         units_lateness_of_lot,
         actual_inventory_carrying_cost,
         priority_value,
         percentage_completion,
         required_completion_time,
         actual_completion_time,
         cpt,
         total_wait_time_in_queues_so_far,
         tpt_completed_so_far,
         time_of_entry_into_the_shop,
         total_required_processing_time,
         time_of_joining_the_queue,
         time_when_processing_starts   : real;
       end;

basic_data_about_an_assly =
       record
          assly_no,
          assly_no_after_processing,
          parent_no,
          no_of_child_asslys,
          no_of_children_for_assembly,
          no_of_processes_needed,
          qty_for_parent,
          assly_on_mc_no  : integer;

          actual_inventory_carrying_cost,
          theoretical_inventory_carrying_cost,
          total_setup_cost,
          cost : real;

          lot_sizing_to_be_done,
          immediately_gets_into_assly,
          immediately_needs_processing : boolean;

          aps
                 : array[1..max_no_of_proces_on_any_assly * 3]
                         of process_data;

          array_of_child_assly
                 : array[1..max_subassly_no_into_an_assly]
                         of child_assly;
       end;
```

```
        lot_data_about_an_assly =
               record
                  assly_no,
                  lot_no_which_will_enter_the_shop,
                  no_of_lots : integer;

                  dol : array[1..max_no_of_lots] of lot;
               end;

        general_machine = record
                            tpt,
                            total_idle_time,
                            cu   : real;
                          end;

        assly_machine = record
                          finished_assly_no : integer;

                          processing_time_per_unit,
                          tpt,
                          total_idle_time,
                          cu   : real;
                        end;

        event_record = record
                         event : event_type;

                         array_of_relevant_nos : array[1..3] of integer;
                           (* 1 - assly_no
                              2 - lot_no
                              3 - mc_no_to_visit *)

                         execution_time : real;

                         next_event : ptr_to_event;
                       end;

(* BEGIN TYPES FOR EXPLOSION DETAILS *)

explosion_details = record
                      assly_no,
                      qty_to_be_exploded : integer;
                    end;

(* END TYPES FOR EXPLOSION DETAILS *)

(* BEGIN TYPES FOR QUEUES *)

lot_and_assly_no  = record
                      lot_no,
                      assly_no : integer;
                    end;

type_for_queue_before_general_mc
                  = record
                      queue_length    : integer;
                      details_of_queue : array[1..max_queue_length]
                                           of lot_and_assly_no;
```

```
                    end;

type_for_queue_before_assly_mc =
             record
                finish_assly_no,
                no_of_subasslys_into_assly : integer;

                quantity_available : array[1..max_subassly_no_into_an_assly]
                                        of integer;

                details_of_queues : array[1..max_subassly_no_into_an_assly]
                                        of type_for_queue_before_general_mc;
             end;

(* END TYPES FOR QUEUES *)

(* BEGIN TYPES FOR WAGNER WHITIN ALGORITHM *)

type   basic_period_data = record
                            order_cost,
                            holding_cost : real;
                            demand : integer;
                           end;

 input_data_for_lot_sizing = array[1..max_period_no] of basic_period_data;

 order_data = record
               period_no,
               qty_produced : integer;
              end;

computation_data = record
                    cost: real;
                    order_period_no : integer;
                   end;

output_data_of_lot_sizing =
       record
         no_of_orders_placed : integer;
         order_schedule : array[1..max_period_no] of order_data;
         total_cost,
         holding_cost,
         order_cost : real;
       end;

period_wise_computations =
       record
        no_of_calculations,
        min_cost_position : integer;

        period_wise_calculation :
          array[1..computing_points_per_period_in_w_w] of
            computation_data;
       end;

type_for_find_min_position = array[1..max_period_no] of real;

(* END TYPES FOR WAGNER WHITIN ALGORITHM *)
```

```
var     clock_time      :real;

        demand,
        exploded_demand : array[1..max_assly_no,1..max_period_no] of integer;
        event_freetop,
        ptr_to_current_executable_event,
        top_of_event_list,
        new_event,
        p               : ptr_to_event;

        abda
            : array[1..max_assly_no] of basic_data_about_an_assly;

        ala
            : array[1..max_assly_no] of lot_data_about_an_assly;

        agm :
              array[1..max_mc_no] of general_machine;
        aam :
              array[range_of_assly_mcs] of assly_machine;

        list_of_end_products : array[1..max_assly_no] of integer;
(*BEGIN VARIABLES FOR EXPLOSION DETAILS *)
        no_of_asslys_on_explosion_list,
        no_of_end_products : integer;

        incremental_explosion_list :
           array[1..(50*max_assly_no)] of explosion_details;
(* END VARIABLES FOR EXPLOSION DETAILS *)

(* BEGIN VARIABLES FOR QUEUES *)

queue_before_general_mc : array[1..max_mc_no] of
                              type_for_queue_before_general_mc;

queue_before_assly_mc   : array[range_of_assly_mcs] of
                              type_for_queue_before_assly_mc;

(* END VARIABLES FOR QUEUES *)

(* BEGIN VARIABLES FOR WAGNER WHITIN *)

input_data_for_w_w : input_data_for_lot_sizing;

      output_data_of_w_w : output_data_of_lot_sizing;

      computation_table_for_w_w: array[1..max_period_no] of
                                    period_wise_computations;

      array_for_output_of_w_w : array[1..max_assly_no] of
                                        output_data_of_lot_sizing;
      in_ass_dat:text;
      in_dem_dat:text;
      in_ass_mc :text;
      w_w_output:text;
      w_w_file : text;
      event_file :text;
```

```
      potty_output:text;
      s_dat:text;
      i,j  :integer;
      counter:integer;
      in_out :text;
      rand   :real;
      flag   :boolean;
      mc_busy:array[1..max_no_of_proces_on_any_assly] of boolean;

(* END VARIABLES FOR WAGNER WHITIN *)
(* start variables for potty's algo *)
R,K,S:array[1..max_assly_no] of integer;
theoretical_cost,actual_cost:array[1..max_assly_no] of real;
best_sol:real;
count,max_count:integer;
stop,go_ahead:boolean;
total_units_lateness:real;
temp,max_dif:real;
index,period:integer;
old_orders,new_orders:integer;
assly_after_processing:integer;
iteration:integer;
levels:integer;
coordination_index,
present_coordination_index,
ag_coordination_index      :array[1..max_assly_no] of real;
max_time,max_index,
present_mean_time,
prev_mean_time      :array[range_of_assly_mcs] of real;
critical_child :array[1..max_assly_no] of integer;
times:integer;
arrival_time:array[1..100,1..max_assly_no,1..max_no_of_lots] of real;
check_out:text;
a,b,c,lots:integer;
assly_count:array[range_of_assly_mcs] of integer;
time_of_reaching        :array[1..max_assly_no,1..max_no_of_lots] of real;
latest_assly_time :array[1..max_assly_no,range_of_assly_mcs] of real;
first_run  :boolean;
earliest_lot_time,
latest_lot_time   :array[range_of_assly_mcs] of real;
prev_range:real;


procedure    event_addfreelist(p : ptr_to_event);

begin (* event_addfreelist *)
    p^.next_event:=event_freetop;
    event_freetop:=p;
end; (* event_addfreelist *)

procedure   newmodified(var p : ptr_to_event);

begin (* newmodified *)
    if (event_freetop = nil)
    then new(p)
    else begin
            p:=event_freetop;
            event_freetop:=event_freetop^.next_event;
```

```
        end;
    p^.next_event:=nil;
    p^.array_of_relevant_nos[1]:=0;
    p^.array_of_relevant_nos[2]:=0;
    p^.execution_time:=0.0;
end; (* newmodified *)


procedure   print_the_event_list;   (* this is the changed version *)

var
       q : ptr_to_event;


begin (* print_the_event_list *)
      q:=ptr_to_current_executable_event;
      case q^.event of
        collect_statistics_from_now_on :
           write(event_file,'collect_statistics_from_now_on ');
        lot_enters_the_shop :
           write(event_file,'lot_enters_the_shop ');
        lot_enters_the_general_mc_queue :
           write(event_file,'lot_enters_the_general_mc_queue ');
        lot_enters_the_assly_mc_queue :
           write(event_file,'lot_enters_the_assly_mc_queue ');
        processing_on_general_mc_starts :
           write(event_file,'processing_on_general_mc_starts ');
        processing_on_assly_mc_starts :
           write(event_file,'processing_on_assly_mc_starts ');
        processing_on_general_mc_gets_over :
           write(event_file,'processing_on_general_mc_gets_over ');
        processing_on_assly_mc_gets_over :
           write(event_file,'processing_on_assly_mc_gets_over ');
        lot_enters_the_finished_product_list:
           write(event_file,'lot_enters_the_finished_product_list');
      end; (* case *)

      write(event_file,'assly_no:',q^.array_of_relevant_nos[1]:1,' ',
                       'lot_no:',q^.array_of_relevant_nos[2]:1,' ');
      writeln(event_file,'exec_time',q^.execution_time:8:3);
end; (* print_the_event_list *)


procedure open_print_the_event_list;

begin
    rewrite(event_file,'event.file');
end;

procedure close_print_the_event_list;

begin
      close(event_file);
end;


procedure   lot_sizing_by_w_w(assly_no : integer);
```

```
var   period_no,
      order_can_be_placed_from_period_no:integer;

procedure   forward_recursion_of_w_w;

var    period_no,
       index,
       min_pos,
       scratch,
       size,
       no_of_calculations_done,
       search_index_in_previous_period,
       min_cost_pos_upto_last_period,
       order_placed_in_period : integer;

       min_cost_upto_last_period,
       cost,
       min_cost_upto_period_in_which_order_is_placed : real;

       a : type_for_find_min_position;

procedure     find_min_position(a:type_for_find_min_position;
                                size: integer;
                                var min_pos : integer);

var    index : integer;

       min_cost : real;

begin (* find_min_position *)
       min_cost:=a[1];
       min_pos:=1;
       for index:=2 to size do
       if (a[index] < min_cost)
       then begin
              min_cost:=a[index];
              min_pos:=index;
            end;
end; (* find_min_position *)

begin (* forward_recursion_of_w_w *)
  computation_table_for_w_w[1].no_of_calculations:=1;
  computation_table_for_w_w[1].min_cost_position:=1;
  computation_table_for_w_w[1].period_wise_calculation[1].cost:=
    input_data_for_w_w[1].order_cost;
  computation_table_for_w_w[1].period_wise_calculation[1].
    order_period_no:=1;
  order_can_be_placed_from_period_no :=1;
  for period_no:=2 to max_period_no do
  begin
    no_of_calculations_done:=1;
    computation_table_for_w_w[period_no].
       no_of_calculations:=period_no - order_can_be_placed_from_period_no+1;
    order_placed_in_period:=period_no;
    min_cost_pos_upto_last_period:=
     computation_table_for_w_w[period_no-1].min_cost_position;
    min_cost_upto_last_period:=
     computation_table_for_w_w[period_no-1].
```

```
     period_wise_calculation[min_cost_pos_upto_last_period].cost;
    computation_table_for_w_w[period_no].
      period_wise_calculation[1].cost:=
     min_cost_upto_last_period +
      input_data_for_w_w[period_no].order_cost;
    computation_table_for_w_w[period_no].
      period_wise_calculation[no_of_calculations_done].
        order_period_no:=order_placed_in_period;
    search_index_in_previous_period:=1;
    order_placed_in_period:=order_placed_in_period - 1;
    while (order_placed_in_period >= order_can_be_placed_from_period_no) do
    begin
      no_of_calculations_done:=no_of_calculations_done + 1;
      min_cost_upto_period_in_which_order_is_placed:=
        computation_table_for_w_w[period_no-1].
          period_wise_calculation[search_index_in_previous_period].cost;
      cost:=min_cost_upto_period_in_which_order_is_placed;
      for index:= order_placed_in_period to (period_no-1) do
      cost:=cost + input_data_for_w_w[index].holding_cost *
                   input_data_for_w_w[period_no].demand;
     computation_table_for_w_w[period_no].
       period_wise_calculation[no_of_calculations_done].cost:=cost;
     computation_table_for_w_w[period_no].
       period_wise_calculation[no_of_calculations_done].
         order_period_no:=order_placed_in_period;
     search_index_in_previous_period:=search_index_in_previous_period  + 1;
     order_placed_in_period:=order_placed_in_period - 1;
    end; (* while *)

    size:= computation_table_for_w_w[period_no].
              no_of_calculations;
    for index:=1 to size do
    a[index]:=computation_table_for_w_w[period_no].
       period_wise_calculation[index].cost;
    find_min_position(a,size,min_pos);
    computation_table_for_w_w[period_no].min_cost_position:=min_pos;
    order_can_be_placed_from_period_no:=
    computation_table_for_w_w[period_no].
      period_wise_calculation[min_pos].
  end; (* for *)

end;(* forward_recursion_of_w_w *)

procedure     fill_up_the_order_schedule;

var       prodn_from_period,
          period_no,
          min_position,
          no_of_orders_placed,
          index,
          order_qty              : integer;

          order_cost,
          total_cost,
          holding_cost           : real;

          computations_to_continue : boolean;
```

```
begin (* fill_up_the_order_schedule *)

    min_position:=computation_table_for_w_w[max_period_no].
                        min_cost_position;
    total_cost:=computation_table_for_w_w[max_period_no].
                        period_wise_calculation[min_position].cost;
    order_cost:=0.0;
    no_of_orders_placed:=0;
    period_no:=max_period_no;
    computations_to_continue:=true;
    while (computations_to_continue) do
    begin
      min_position:=computation_table_for_w_w[period_no].
                        min_cost_position;
      prodn_from_period:=computation_table_for_w_w[period_no].
                            period_wise_calculation[min_position].
                               order_period_no;
      order_cost:=order_cost+input_data_for_w_w[prodn_from_period].order_cost;
      no_of_orders_placed:=no_of_orders_placed + 1;
      order_qty:=0;
      for index:=prodn_from_period to period_no do
            order_qty:=order_qty + input_data_for_w_w[index].demand;
      output_data_of_w_w.order_schedule[no_of_orders_placed].period_no:=
         prodn_from_period;
      output_data_of_w_w.order_schedule[no_of_orders_placed].qty_produced:=
         order_qty;
      period_no:=prodn_from_period - 1;
      if (period_no = 0) then computations_to_continue:=false;
    end;
    holding_cost:=total_cost - order_cost ;
    output_data_of_w_w.no_of_orders_placed:=no_of_orders_placed;
    output_data_of_w_w.total_cost:=total_cost;
    output_data_of_w_w.holding_cost:=holding_cost;
    output_data_of_w_w.order_cost:=order_cost;

end; (* fill_up_the_order_schedule *)

procedure   print_w_w_output;

var     index,
        period_no,
        no_of_items_printed_so_far,
        no_of_items_to_be_printed,
        no_of_calculations_done : integer;


begin (* print_w_w_output *)
   writeln(w_w_output,'assembly no.= ',assly_no:2);
   for period_no:=1 to max_period_no do
   begin
     writeln(w_w_output,'period_no:=',period_no:2,' ');
     no_of_calculations_done:=computation_table_for_w_w[period_no].
                                 no_of_calculations;
     no_of_items_to_be_printed:=no_of_calculations_done;
     no_of_items_printed_so_far:=0;
     while (no_of_items_to_be_printed > 0) do
       if (no_of_items_to_be_printed <= 5)
       then
```

```
      begin
        write(w_w_output,' ':12);
        for index:=(no_of_items_printed_so_far+1)
                              to
                 no_of_calculations_done
                              do
        begin
          write(w_w_output,computation_table_for_w_w[period_no].
                           period_wise_calculation[index].cost:8:3,'(');
          write(w_w_output,computation_table_for_w_w[period_no].
                           period_wise_calculation[index].
                             order_period_no,')');
          no_of_items_to_be_printed:=no_of_items_to_be_printed-1;
        end; (* for index *)
        writeln(w_w_output);
        no_of_items_printed_so_far:=no_of_calculations_done;
      end (* then *)
      else
      begin
        write(w_w_output,' ':12);
        for index:=(no_of_items_printed_so_far+1)
                               to
                 (no_of_items_printed_so_far+5)
                               do
        begin
          write(w_w_output,computation_table_for_w_w[period_no].
                           period_wise_calculation[index].cost:8:3,'(');
          write(w_w_output,computation_table_for_w_w[period_no].
                           period_wise_calculation[index].
                             order_period_no,')');
        end;
        writeln(w_w_output);
        no_of_items_printed_so_far:=no_of_items_printed_so_far+10;
        no_of_items_to_be_printed:=no_of_items_to_be_printed-10;
      end; (* else *)
    end; (* for period_no *)
    writeln(w_w_output);
    writeln(w_w_output);
    writeln(w_w_output);
    writeln(w_w_output,'production_schedule is as follows');
    writeln(w_w_output, 'period_no', 'production_quantity');
    for index:=1 to output_data_of_w_w.no_of_orders_placed do
    writeln(w_w_output,output_data_of_w_w.order_schedule[index].
                            period_no:8,' ':5,
                       output_data_of_w_w.order_schedule[index].
                            qty_produced);
    writeln(w_w_output,'order_cost:=',output_data_of_w_w.order_cost);
    writeln(w_w_output,'holding_cost:=',output_data_of_w_w.holding_cost);
    writeln(w_w_output,'total_cost:=',output_data_of_w_w.total_cost);
end; (* print_w_w_output *)

begin (* lot_sizing_by_w_w *)

  forward_recursion_of_w_w;

  fill_up_the_order_schedule;

(*  print_w_w_output;*)
```

```
end; (* lot_sizing_by_w_w *)


procedure     read_assly_data;

var

        assembly_no,
        child_assly_no,
        needs_process,
        needs_assembly,
        process_no      : integer;

begin (* read_assly_data *)
      reset(in_ass_dat,'in_ass.dat');
      for assembly_no:=1 to max_assly_no do
      begin
          readln(in_ass_dat,
              abda[assembly_no].assly_no,
              abda[assembly_no].
              no_of_child_asslys,needs_process,needs_assembly,
              abda[assembly_no].cost);
          abda[assembly_no].
          immediately_needs_processing:=(needs_process = 1);
          abda[assembly_no].
          immediately_gets_into_assly:= (needs_assembly=1);
          if (abda[assembly_no].
                 no_of_child_asslys > 0)
          then
            for child_assly_no:=1 to
               abda[assembly_no].
                                          no_of_child_asslys do
            readln(in_ass_dat,
               abda[assembly_no].
               array_of_child_assly[child_assly_no].assly_no,
               abda[assembly_no].
               array_of_child_assly[child_assly_no].qty_to_be_contributed);
          if(abda[assembly_no].immediately_needs_processing)
          then
            begin
              readln(in_ass_dat,
                 abda[assembly_no].
                    no_of_children_for_assembly,
                 abda[assembly_no].
                                 assly_no_after_processing,
                 abda[assembly_no].
                                 no_of_processes_needed);
              for process_no:=1 to
                  abda[assembly_no].
                                  no_of_processes_needed do
              begin (* for *)
                with abda[assembly_no].
                        aps[process_no] do
                begin (* with *)
                  readln(in_ass_dat,
                             mc_no,setup_time,processing_time_per_unit,
                             setup_cost);
```

```
                end; (* with *)
              end; (* for *)
            end
          else (* collect_parent_data *)
            begin
              abda[assembly_no].
              no_of_processes_needed:=0;
              readln(in_ass_dat,
                 abda[assembly_no].parent_no,
                 abda[assembly_no].
                                   qty_for_parent,
                 abda[assembly_no].
                                   assly_on_mc_no);
            end;
      end; (* for *)

end; (* read_assly_data *)

procedure    read_assly_demands;

var

         assly_no,
         period_no,
         no_of_demands,
         demand_no    : integer;

begin (* read_assly_demands *)

      for assly_no:=1 to max_assly_no do
      for period_no:=1 to max_period_no do
      demand[assly_no,period_no]:=0;
      reset(in_dem_dat,'in_dem.dat');
      while not eof(in_dem_dat) do
      begin
        readln(in_dem_dat,assly_no,no_of_demands);
        for demand_no:=1 to no_of_demands do
        begin
          read(in_dem_dat,period_no);
          readln(in_dem_dat,demand[assly_no,period_no]);
        end;
      end;

end; (* read_assly_demands *)

procedure     read_input_data;

begin (* read_input_data *)

      read_assly_data;
      read_assly_demands;

end; (* read_input_data *)

procedure      transfer_asslys_on_explosion_list(period_no : integer);

var    assly_no : integer;
```

```
begin (* transfer_asslys_on_explosion_list *)
     no_of_asslys_on_explosion_list:=0;
     for assly_no:=1 to max_assly_no do
     if (demand[assly_no,period_no] > 0)
     then
         begin
           no_of_asslys_on_explosion_list:=
           no_of_asslys_on_explosion_list + 1;
            incremental_explosion_list[no_of_asslys_on_explosion_list].
                    assly_no:=assly_no;
           incremental_explosion_list[no_of_asslys_on_explosion_list].
                    qty_to_be_exploded:=
           demand[assly_no,period_no];
         end;

end; (* transfer_asslys_on_explosion_list *)

procedure       do_explosion;

var      current_no,
         current_assly_no,
         current_qty_for_explosion,
         no_of_children,
         children_no,
         ending_no,
         period_no,
         temp_assly,
         temp_qty,
         a,b             : integer;

begin (* do_explosion *)
     for a:= 1 to max_assly_no do
     for b:= 1 to max_period_no do
     exploded_demand[a,b]:=0;
     for period_no:=1 to max_period_no do
     begin
       transfer_asslys_on_explosion_list(period_no);
       current_no:=1;
       ending_no:=no_of_asslys_on_explosion_list;
       while (current_no <= ending_no) do
       begin
          current_assly_no:=incremental_explosion_list[current_no]
                                                          .assly_no;
          current_qty_for_explosion:=
                       incremental_explosion_list[current_no].
                                                qty_to_be_exploded;
          exploded_demand[current_assly_no,period_no]:=
              exploded_demand[current_assly_no,period_no]
                              +current_qty_for_explosion;
          no_of_children:=
             abda[current_assly_no].
                                     no_of_child_asslys;
          for children_no:=1 to no_of_children do
          begin
             no_of_asslys_on_explosion_list:=
                no_of_asslys_on_explosion_list + 1;
             temp_assly:=abda[current_assly_no].
```

```
                array_of_child_assly[children_no].assly_no;
            temp_qty:=abda[current_assly_no].
              array_of_child_assly[children_no].qty_to_be_contributed
                   * current_qty_for_explosion;
            exploded_demand[temp_assly,period_no]:=
            exploded_demand[temp_assly,period_no] + current_qty_for_explosion;
          end;
          current_no:=current_no + 1;
        end;
      end;

end; (* do_explosion *)


procedure     do_lot_sizing;

var    setup_cost : real;

       assly_no,
       assly_no_after_processing,
       period_no,
       process_no  : integer;


begin (* do_lot_sizing *)

       rewrite(w_w_output,'w_w.out');
   for assly_no:=1 to max_assly_no do
   abda[assly_no].
        lot_sizing_to_be_done:=true;
   for assly_no:=1 to max_assly_no do
   begin (* for *)
     if (abda[assly_no].
                    immediately_needs_processing)
     then
     begin (* then *)
       assly_no_after_processing:=
       abda[assly_no].
             assly_no_after_processing;
       abda[assly_no_after_processing].
             lot_sizing_to_be_done:=false;
     end; (* then *)
   end; (* for *)

       for assly_no:=1 to max_assly_no do
       if (abda[assly_no].
                lot_sizing_to_be_done)
       then
       begin (* then *)
          for period_no:=1 to max_period_no do
          input_data_for_w_w[period_no].holding_cost:=
             abda[assly_no].cost*
          inventory_cost_fraction;
          setup_cost:=0.0;
          for process_no:=1 to
                   abda[assly_no].
                                        no_of_processes_needed do
          setup_cost:=setup_cost +
```

```
            abda[assly_no].
              aps[process_no].setup_cost;
          abda[assly_no].total_setup_cost:=
          setup_cost;
          for period_no:=1 to max_period_no do
          input_data_for_w_w[period_no].order_cost:=
             setup_cost;
          for period_no:=1 to max_period_no do
          input_data_for_w_w[period_no].demand:=
            exploded_demand[assly_no,period_no];
          lot_sizing_by_w_w(assly_no);
          array_for_output_of_w_w[assly_no]:=output_data_of_w_w;
          if (abda[assly_no].
                       immediately_needs_processing)
          then
          begin (* then *)
            assly_no_after_processing:=
            abda[assly_no].
                   assly_no_after_processing;
            array_for_output_of_w_w[assly_no_after_processing]:=
                   output_data_of_w_w;
          end; (* then *)
        end; (* then *)
        close(w_w_output);

end; (* do_lot_sizing *)



procedure   prepare_queue_of_lots;

var    (* lead_time, *)
          assly_no,
          no_of_lots,
          process_no,
          lot_no       : integer;

begin (* prepare_queue_of_lots *)

      for assly_no:=1 to max_assly_no do
      begin
        ala[assly_no].assly_no:=assly_no;
        ala[assly_no].no_of_lots:=
          array_for_output_of_w_w[assly_no].no_of_orders_placed;
        no_of_lots:=array_for_output_of_w_w[assly_no].
                 no_of_orders_placed;
        for lot_no:=1 to no_of_lots do
        begin
          ala[assly_no].
              dol[lot_no].assly_no:=assly_no;
          ala[assly_no].
              dol[lot_no].original_size:=
          array_for_output_of_w_w[assly_no].
              order_schedule[lot_no].qty_produced;
          ala[assly_no].
              dol[lot_no].qty_left:=
          array_for_output_of_w_w[assly_no].
              order_schedule[lot_no].qty_produced;
```

```
ala[assly_no].
   dol[lot_no].
    no_of_agm_visited:=0;
ala[assly_no].
   dol[lot_no].
    current_mc_no:=0;
ala[assly_no].
   dol[lot_no].
    processing_complete:=false;
ala[assly_no].
   dol[lot_no].
     actual_inventory_carrying_cost:=0.0;
ala[assly_no].
   dol[lot_no].
     priority_value:=0.0;
ala[assly_no].
   dol[lot_no].
     percentage_completion:=0.0;
ala[assly_no].
   dol[lot_no].
     required_completion_time:=
array_for_output_of_w_w[assly_no].
     order_schedule[lot_no].period_no*
                 no_of_hours_in_a_period;
ala[assly_no].
   dol[lot_no].
     actual_completion_time:=0.0;
ala[assly_no].
   dol[lot_no].
     cpt:=0.0;
ala[assly_no].
   dol[lot_no].
     total_wait_time_in_queues_so_far:=0.0;
ala[assly_no].
   dol[lot_no].
     tpt_completed_so_far:=0.0;
ala[assly_no].
   dol[lot_no].
    time_of_entry_into_the_shop:=
array_for_output_of_w_w[assly_no].
    order_schedule[lot_no].period_no*1.0 - lead_time;
ala[assly_no].
   dol[lot_no].
    total_required_processing_time:=0.0;

for process_no:=1 to abda[assly_no].
                              no_of_processes_needed do
begin
  ala[assly_no].
    dol[lot_no].
      total_required_processing_time:=
  ala[assly_no].
    dol[lot_no].
      total_required_processing_time +
  abda[assly_no].
    aps[process_no].
       setup_time +
  abda[assly_no].
```

```
              aps[process_no].
                 processing_time_per_unit *
            array_for_output_of_w_w[assly_no].
              order_schedule[lot_no].qty_produced;
          end;
          ala[assly_no].
             dol[lot_no].
              time_of_joining_the_queue:=0.0;
        end;
      end;

end;(* prepare_queue_of_lots *)


procedure   plug_in_the_event(p : ptr_to_event);

var     execution_time_just_greater,
        prev_to_execution_time_just_greater : ptr_to_event;

begin (* plug_in_the_event *)
    counter:=counter + 1;
    if (top_of_event_list = nil)
    then
      begin (* top_of_event_list = nil *)
        top_of_event_list:= p;
        p:=nil;
      end (* top_of_event_list = nil *)
    else
      begin (* contains at least one event *)
        execution_time_just_greater:=top_of_event_list;
        prev_to_execution_time_just_greater:=top_of_event_list;

        if (top_of_event_list^.execution_time < p^.execution_time)
        then
          begin (* continue search till execution_time_just_greater *)
            execution_time_just_greater:=
              execution_time_just_greater^.next_event;

            while (execution_time_just_greater <> nil)
                                  and
                 (execution_time_just_greater^.
                         execution_time < p^.execution_time)
                                  do
            begin (* while *)
              execution_time_just_greater:=
                execution_time_just_greater^.next_event;
              prev_to_execution_time_just_greater:=
                prev_to_execution_time_just_greater^.next_event;
            end; (* while *)

            if (execution_time_just_greater <> nil)
            then
               begin
                 p^.next_event:=
                 prev_to_execution_time_just_greater^.next_event;
                 prev_to_execution_time_just_greater^.next_event:=p;
                 p:=nil;
               end
```

```
            else
               begin
                 prev_to_execution_time_just_greater^.next_event:=p;
                 p:=nil;
               end;
          end (* continue search till execution_time_just_greater *)
        else
        begin (* execution time just greater *)
          p^.next_event:=top_of_event_list;
          top_of_event_list:=p;
          p:=nil;
        end; (* execution time just greater *)
      end; (* contains at least one event *)

end; (* plug_in_the_event *)

procedure   prepare_events_regarding_entry_of_components;

var     assly_no,
        no_of_lots,
        lot_no        : integer;

begin (* prepare_events_regarding_entry_of_components *)

    for assly_no:=1 to max_assly_no do
    if (abda[assly_no].
            no_of_child_asslys = 0)
    then
    begin (* then it is a component assly *)
      no_of_lots:=ala[assly_no].no_of_lots;
      for lot_no:=1 to no_of_lots do
      begin (* for lot_no *)
        newmodified(p);
        with p^ do
        begin
          event:=lot_enters_the_shop;
          array_of_relevant_nos[1]:=assly_no;
          array_of_relevant_nos[2]:=lot_no;
          execution_time:=ala[assly_no].
                                dol[lot_no].
                                  time_of_entry_into_the_shop;
          plug_in_the_event(p);
        end;
      end; (* for lot_no *)
    end; (* then it is a component assly *)

end; (* prepare_events_regarding_entry_of_components *)

procedure    prepare_collect_statistic_event;

begin (* prepare_collect_statistic_event *)

        newmodified(p);
        with p^ do
        begin
          event:=collect_statistics_from_now_on;
          array_of_relevant_nos[1]:=0;
          array_of_relevant_nos[2]:=0;
```

```
          execution_time:=collect_statistics_from_time;
          plug_in_the_event(p);
        end;

end; (* prepare_collect_statistic_event *)

procedure    initialisations;

var    mc_no,
       assly_mc_no,
       assly_no_after_processing,
       finished_assly_no,
       assly_no,
       subassly_no,
       quelg_no   : integer;

       in_ass_mc : text;

begin (* initialisations *)

       clock_time:=0.0;
       counter:=0;
       flag:=false;

(* now all mc queues and assly mc queues are initialised *)

       for mc_no:=1 to max_mc_no do
       begin
         queue_before_general_mc[mc_no].queue_length:=0;
         for quelg_no:=1 to max_queue_length do
         begin
           queue_before_general_mc[mc_no].
                details_of_queue[quelg_no].lot_no:=0;
           queue_before_general_mc[mc_no].
                details_of_queue[quelg_no].assly_no:=0;
         end;
       end;
       reset(in_ass_mc,'in_ass.mc');
       for assly_mc_no:=16 to 20 do
       begin
         readln(in_ass_mc,
                aam[assly_mc_no].
                                finished_assly_no,
                aam[assly_mc_no].
                                 processing_time_per_unit,
                queue_before_assly_mc[assly_mc_no].
                                    no_of_subasslys_into_assly );
         finished_assly_no:=
         aam[assly_mc_no].
                                           finished_assly_no;
   (*      queue_before_assly_mc[assly_mc_no].
                no_of_subasslys_into_assly:=
         abda[finished_assly_no].
                no_of_child_asslys;          *)
         for subassly_no:=1 to max_subassly_no_into_an_assly do
         begin
           queue_before_assly_mc[assly_mc_no].
                details_of_queues[subassly_no].queue_length:=0;
```

```
                quantity_available[subassly_no]:=0;(* added *)
            for quelg_no:=1 to max_queue_length do
            begin
              queue_before_assly_mc[assly_mc_no].
                       details_of_queues[subassly_no].
                          details_of_queue[quelg_no].lot_no:=0;
              queue_before_assly_mc[assly_mc_no].
                       details_of_queues[subassly_no].
                          details_of_queue[quelg_no].assly_no:=0;
            end;
          end;
        end;

(* now all mc queues and assly mc queues are initialised *)

(* now we do all pointer initialisations *)

    event_freetop:=nil;
    ptr_to_current_executable_event:=nil;
    top_of_event_list:=nil;
    p:=nil;

(* pointer initialisations are complete *)

(* initialising general machine data *)

   for mc_no:=1 to max_mc_no do
   with agm[mc_no] do
   begin (* with *)
     tpt:=0.0;
     total_idle_time:=0.0;
     cu:=0.0;
   end; (* with *)

(* end of initialising general machine data *)

(* initialising assembly machine data *)

   for assly_mc_no:=16 to 20 do
   with aam[assly_mc_no] do
   begin (* with *)
     tpt:=0.0;
     total_idle_time:=0.0;
     cu:=0.0;
   end; (* with *)

(* end of initialising assembly machine data *)

(* lot size needed or not is determined *)

   for assly_no:=1 to max_assly_no do
   abda[assly_no].
        lot_sizing_to_be_done:=true;
   for assly_no:=1 to max_assly_no do
   begin (* for *)
     if (abda[assly_no].
                    immediately_needs_processing)
```

```
      then
      begin (* then *)
        assly_no_after_processing:=
        abda[assly_no].
              assly_no_after_processing;
        abda[assly_no_after_processing].
              lot_sizing_to_be_done:=false;
      end; (* then *)
    end; (* for *)
    for assly_no:=1 to max_assly_no do
    ala[assly_no].
    lot_no_which_will_enter_the_shop:=1;
    for mc_no:=1 to max_no_of_proces_on_any_assly do
    mc_busy[mc_no]:=false;

end;(* initialisations *)

procedure       find_the_next_executable_event;

begin (* find_the_next_executable_event *)

        ptr_to_current_executable_event:=top_of_event_list;
        if (ptr_to_current_executable_event=nil) then
          flag:=true
        (* no event left *)
        else
        begin
           top_of_event_list:=top_of_event_list^.next_event;
           clock_time:=ptr_to_current_executable_event^.
                                execution_time;
        end;
end; (* find_the_next_executable_event *)

procedure     search_and_insert_for_general_mc_queue(mc_no : integer);

var    i,j,k,n,index,
       insert_position,
       assly_no,
       lot_no,
       current_assly_no,
       current_lot_no,
       next_assly_no,
       assly_mc_no        : integer;

       priority_value_of_current_lot : real;

       insert_position_found : boolean;

begin (* search_and_insert_for_general_mc_queue *)
     current_assly_no:=ptr_to_current_executable_event^.
                             array_of_relevant_nos[1];
     current_lot_no:=ptr_to_current_executable_event^.
                             array_of_relevant_nos[2];
    if (queue_before_general_mc[mc_no].queue_length = 0)
    then begin
           queue_before_general_mc[mc_no].
                details_of_queue[1].assly_no:=
           current_assly_no;
```

```
        queue_before_general_mc[mc_no].
             details_of_queue[1].lot_no:=
        current_lot_no;
        queue_before_general_mc[mc_no].queue_length:=
        queue_before_general_mc[mc_no].queue_length + 1;
      end
else
begin
  i:=1;j:=queue_before_general_mc[mc_no].queue_length;
  current_assly_no:=ptr_to_current_executable_event^.
                        array_of_relevant_nos[1];
  current_lot_no:=ptr_to_current_executable_event^.
                        array_of_relevant_nos[2];
  next_assly_no:=abda[current_assly_no].assly_no_after_processing;
  assly_mc_no:=abda[next_assly_no].assly_on_mc_no;
  if not first_run then
  coordination_index[current_assly_no]:=
     1 + (latest_assly_time[current_assly_no,assly_mc_no] -
           prev_mean_time[assly_mc_no])*0.1;
  priority_value_of_current_lot:=
  ala[current_assly_no].
     dol[current_lot_no].priority_value *
               coordination_index[current_assly_no];
  insert_position_found:=false;

  repeat
     k:=(i+j) div 2;
     assly_no:=
     queue_before_general_mc[mc_no].details_of_queue[k].assly_no;
     lot_no:=
     queue_before_general_mc[mc_no].details_of_queue[k].lot_no;

     if (priority_value_of_current_lot >
           ala[assly_no].
              dol[lot_no].priority_value)
     then begin (* then *)
            if (k=1)
            then begin
                   insert_position_found:=true;
                   insert_position:=1;
                 end
            else
            begin
              assly_no:=
              queue_before_general_mc[mc_no].
                       details_of_queue[k-1].assly_no;
              lot_no:=
              queue_before_general_mc[mc_no].
                       details_of_queue[k-1].lot_no;
              if (priority_value_of_current_lot <=
                        ala[assly_no].
                            dol[lot_no].priority_value)
              then begin
                       insert_position_found:=true;
                       insert_position:=k;
                     end
              else j:=k-1;
            end;
```

```
            end (* then *)
        else begin (* else *)
               if (k=queue_before_general_mc[mc_no].queue_length)
               then begin
                      insert_position_found:=true;
                      insert_position:=
                         queue_before_general_mc[mc_no].
                                queue_length + 1;
                    end
               else
               begin
                 assly_no:=
                 queue_before_general_mc[mc_no].
                         details_of_queue[k+1].assly_no;
                 lot_no:=
                 queue_before_general_mc[mc_no].
                         details_of_queue[k+1].lot_no;
                 if (priority_value_of_current_lot >=
                           ala[assly_no].
                               dol[lot_no].priority_value)
                 then begin
                         insert_position_found:=true;
                         insert_position:=k+1;
                      end
                 else i:=k+1;
               end;
             end; (* else *)
     until (insert_position_found);

     for index:=queue_before_general_mc[mc_no].queue_length
                                downto
                         insert_position
                                 do
     queue_before_general_mc[mc_no].
                 details_of_queue[index+1]:=
     queue_before_general_mc[mc_no].
                 details_of_queue[index];
     queue_before_general_mc[mc_no].
          details_of_queue[insert_position].assly_no:=
     current_assly_no;
     queue_before_general_mc[mc_no].
          details_of_queue[insert_position].lot_no:=
     current_lot_no;
     queue_before_general_mc[mc_no].queue_length:=
     queue_before_general_mc[mc_no].queue_length + 1;
   end;

end; (* search_and_insert_for_general_mc_queue *)

procedure      search_and_insert_for_assembly_mc_queue(mc_no : integer);

var   finish_assly_no,
      current_assly_no,
      current_lot_no,
      que_len,
      upper_search_lim,
      search_no  : integer;
```

```
      position_found : boolean;

begin (* search_and_insert_for_assembly_mc_queue *)

    current_assly_no:=ptr_to_current_executable_event^.
                              array_of_relevant_nos[1];
    current_lot_no:=ptr_to_current_executable_event^.
                              array_of_relevant_nos[2];
    finish_assly_no:=aam[mc_no].finished_assly_no;

    upper_search_lim:=abda[finish_assly_no].
                          no_of_children_for_assembly;
    search_no:=1; position_found:=false;

    while (search_no <= upper_search_lim) and not (position_found) do
    if (abda[finish_assly_no].
             array_of_child_assly[search_no].assly_no =
        current_assly_no)
    then position_found:=true
    else search_no:=search_no + 1;

    if not(position_found)
    then begin
           writeln('assly_no not found in insert queue for assly mc');
           halt;
         end
    else (* position has been found *)
    begin (* else *)
      queue_before_assly_mc[mc_no].details_of_queues[search_no].
             queue_length:=
      queue_before_assly_mc[mc_no].details_of_queues[search_no].
             queue_length + 1;
      que_len:=
      queue_before_assly_mc[mc_no].details_of_queues[search_no].
             queue_length;
      queue_before_assly_mc[mc_no].details_of_queues[search_no].
             details_of_queue[que_len].assly_no:=current_assly_no;
      queue_before_assly_mc[mc_no].details_of_queues[search_no].
             details_of_queue[que_len].lot_no:=current_lot_no;
      queue_before_assly_mc[mc_no].
             quantity_available[search_no]:=
      queue_before_assly_mc[mc_no].
             quantity_available[search_no] +
      ala[current_assly_no].
             dol[current_lot_no].
                  original_size;
    end; (* else *)

end; (* search_and_insert_for_assembly_mc_queue *)

procedure   execute_collect_statistics_from_now_on;

var     machine_no,
        queue_no,
        no_of_queues,
        queue_length,
        queue_length_index,
        assly_no,
```

```
        lot_no       : integer;

(* collect_statistics_from_time *)

begin (* execute_collect_statistics_from_now_on *)

   for machine_no:=1 to max_mc_no do
   begin (* for general_machine_no *)
     agm[machine_no].tpt:=0.0;
   end; (* for general_machine_no *)
   for machine_no:=16 to 20 do
   begin (* for machine_no *)
     aam[machine_no].tpt:=0.0;
   end; (* for machine_no *)
   for machine_no:=1 to max_mc_no do
   begin (* for general_machine_no *)
     queue_length:=queue_before_general_mc[machine_no].queue_length;
     for queue_length_index:=1 to queue_length do
     begin (* for queue_no *)
       assly_no:=queue_before_general_mc[machine_no].
                        details_of_queue[queue_length_index].assly_no;
       lot_no:=queue_before_general_mc[machine_no].
                        details_of_queue[queue_length_index].lot_no;
      ala[assly_no].
        dol[lot_no].total_wait_time_in_queues_so_far:=0.0;
      ala[assly_no].
      dol[lot_no].time_of_entry_into_the_shop:=
      collect_statistics_from_time;
     end; (* for queue_no *)
   end; (* for general_machine_no *)
   for machine_no:=16 to 20 do
   begin (* for assly_machine_no *)
     no_of_queues:=queue_before_assly_mc[machine_no].
                              no_of_subasslys_into_assly;
     for queue_no:=1 to no_of_queues do
     begin (* for queue_no *)
       queue_length:=queue_before_assly_mc[machine_no].
                        details_of_queues[queue_no].queue_length;
       for queue_length_index:=1 to queue_length do
       begin (* for queue_length_index *)
          assly_no:=
          queue_before_assly_mc[machine_no].
             details_of_queues[queue_no].
                 details_of_queue[queue_length_index].assly_no;
          lot_no:=
          queue_before_assly_mc[machine_no].
             details_of_queues[queue_no].
                 details_of_queue[queue_length_index].lot_no;
          ala[assly_no].
             dol[lot_no].total_wait_time_in_queues_so_far:=0.0;
          ala[assly_no].
             dol[lot_no].time_of_entry_into_the_shop:=
          collect_statistics_from_time;
       end; (* for queue_length_index *)
     end; (* for queue_no *)
   end; (* for assly_machine_no *)
end; (* execute_collect_statistics_from_now_on *)
```

```
procedure   execute_lot_enters_the_shop;

var    assly_no,
       mc_no,
       lot_no    : integer;

begin (* execute_lot_enters_the_shop *)

    newmodified(new_event);
    new_event^.array_of_relevant_nos[1]:=
    ptr_to_current_executable_event^.array_of_relevant_nos[1];
    new_event^.array_of_relevant_nos[2]:=
    ptr_to_current_executable_event^.array_of_relevant_nos[2];
    new_event^.execution_time:=
    ptr_to_current_executable_event^.execution_time;
    assly_no:=ptr_to_current_executable_event^.array_of_relevant_nos[1];
    lot_no:=ptr_to_current_executable_event^.array_of_relevant_nos[2];
    if (abda[assly_no].
                  immediately_gets_into_assly)
    then begin
           new_event^.event:=lot_enters_the_assly_mc_queue;
           mc_no:=abda[assly_no].
                        assly_on_mc_no;
           new_event^.array_of_relevant_nos[3]:=mc_no;
         end
    else if (abda[assly_no].
                  immediately_needs_processing)
         then begin
                new_event^.event:=lot_enters_the_general_mc_queue;
                mc_no:=abda[assly_no].
                           aps[1].mc_no;
                new_event^.array_of_relevant_nos[3]:=mc_no;
              end
         else new_event^.event:=lot_enters_the_finished_product_list;
    ala[assly_no].
         dol[lot_no].time_of_entry_into_the_shop
               :=ptr_to_current_executable_event^.execution_time;
    plug_in_the_event(new_event);

end; (* execute_lot_enters_the_shop *)

procedure   execute_lot_enters_the_general_mc_queue;

var    assly_no,
       lot_no,
       queue_lg,
       mc_no_to_visit ,
       process_no       : integer;

begin (* execute_lot_enters_the_general_mc_queue *)

    assly_no:=ptr_to_current_executable_event^.array_of_relevant_nos[1];
    lot_no:=ptr_to_current_executable_event^.array_of_relevant_nos[2];
    mc_no_to_visit:=ptr_to_current_executable_event^.
                        array_of_relevant_nos[3];
    process_no:=ala[assly_no].dol[lot_no].no_of_agm_visited + 1;
    ala[assly_no].
      dol[lot_no].current_mc_no:=mc_no_to_visit;
```

```
    ala[assly_no].
      dol[lot_no].time_of_joining_the_queue:=
        ptr_to_current_executable_event^.execution_time;
    ala[assly_no].
      dol[lot_no].cpt:=
    ala[assly_no].
      dol[lot_no].original_size *
    abda[assly_no].
      aps[process_no].
        processing_time_per_unit
                  +
    abda[assly_no].
      aps[process_no].
        setup_time;
    search_and_insert_for_general_mc_queue(mc_no_to_visit);
    queue_lg:=queue_before_general_mc[mc_no_to_visit].queue_length;
    if (queue_lg = 1) and (mc_busy[mc_no_to_visit]=false) then
         begin
           newmodified(new_event);
           new_event^.event:=processing_on_general_mc_starts;
           new_event^.array_of_relevant_nos[1]:=assly_no;
           new_event^.array_of_relevant_nos[2]:=lot_no;
           new_event^.array_of_relevant_nos[3]:=mc_no_to_visit;
           new_event^.execution_time:=
           ptr_to_current_executable_event^.execution_time;
           plug_in_the_event(new_event);
         end;
end; (* execute_lot_enters_the_general_mc_queue *)

function    processing_can_start_on_assly_mc
                          (mc_no_to_visit : integer) : boolean;

var   processing_can_start : boolean;

      pos_of_lot_no,
      lot_size,
      no_of_children,
      children_index,
      quantity_to_be_contributed,
      finish_assly_no  : integer;

begin (* processing_can_start_on_assly_mc *)

    finish_assly_no := aam[mc_no_to_visit].
                                                  finished_assly_no;
    {queue_before_assly_mc[mc_no_to_visit].finish_assly_no;}
    pos_of_lot_no:=
    ala[finish_assly_no].
        lot_no_which_will_enter_the_shop;
    lot_size:=
    ala[finish_assly_no].
        dol[pos_of_lot_no].original_size;
    no_of_children:=abda[finish_assly_no].
                          no_of_children_for_assembly;
    processing_can_start:=true;
    children_index:=1;
    while  (processing_can_start)
                    and
```

```
          (children_index <= no_of_children)
                  do
    begin (* while *)
      quantity_to_be_contributed:=
      abda[finish_assly_no].
         array_of_child_assly[children_index].qty_to_be_contributed
         * lot_size;
      if (quantity_to_be_contributed >
            queue_before_assly_mc[mc_no_to_visit].
               quantity_available[children_index])
      then processing_can_start:=false
      else children_index:=children_index + 1;
    end; (* while *)
    processing_can_start_on_assly_mc:=processing_can_start;

end; (* processing_can_start_on_assly_mc *)

procedure   execute_lot_enters_the_assly_mc_queue;

var    assly_no,
       finished_assly_no,
       pos_of_lot_no,
       lot_no,
       queue_lg,
       mc_no_to_visit    : integer;

begin (* execute_lot_enters_the_assly_mc_queue *)

    assly_no:=ptr_to_current_executable_event^.array_of_relevant_nos[1];
    lot_no:=ptr_to_current_executable_event^.array_of_relevant_nos[2];
    mc_no_to_visit:=ptr_to_current_executable_event^.
                        array_of_relevant_nos[3];
    assly_count[mc_no_to_visit]:=
              assly_count[mc_no_to_visit] + 1;
    ala[assly_no].
      dol[lot_no].time_of_joining_the_queue:=
        ptr_to_current_executable_event^.execution_time;
    time_of_reaching[assly_no,lot_no]:=
         ptr_to_current_executable_event^.execution_time;
    present_mean_time[mc_no_to_visit]:=
    ((assly_count[mc_no_to_visit]-1)*present_mean_time[mc_no_to_visit] +
    time_of_reaching[assly_no,lot_no])/assly_count[mc_no_to_visit];
    if (time_of_reaching[assly_no,lot_no]>
                                present_mean_time[mc_no_to_visit])
    then
        latest_assly_time[assly_no,mc_no_to_visit]:=
                    time_of_reaching[assly_no,lot_no];
    arrival_time[times+1,assly_no,lot_no]:=
                     time_of_reaching[assly_no,lot_no];

    search_and_insert_for_assembly_mc_queue(mc_no_to_visit);

    if (processing_can_start_on_assly_mc(mc_no_to_visit))
    then
       begin (* then fire the event processing_starts_on_assly_mc *)
          if  (ptr_to_current_executable_event^.execution_time<
                              earliest_lot_time[mc_no_to_visit]) then
           earliest_lot_time[mc_no_to_visit]:=
```

```
                ptr_to_current_executable_event^.execution_time
            else if
            (ptr_to_current_executable_event^.execution_time>
                            latest_lot_time[mc_no_to_visit]) then
            latest_lot_time[mc_no_to_visit]:=
              ptr_to_current_executable_event^.execution_time;
            finished_assly_no:=
            abda[assly_no].parent_no;
            newmodified(new_event);
            new_event^.event:=processing_on_assly_mc_starts;
            new_event^.array_of_relevant_nos[1]:=finished_assly_no;
            pos_of_lot_no:=
            ala[finished_assly_no].
                     lot_no_which_will_enter_the_shop;
            new_event^.array_of_relevant_nos[2]:=pos_of_lot_no;
            new_event^.array_of_relevant_nos[3]:=mc_no_to_visit;
            new_event^.execution_time:=
            ptr_to_current_executable_event^.execution_time;
            plug_in_the_event(new_event);
        end; (* then fire the event processing_starts_on_assly_mc *)

end; (* execute_lot_enters_the_assly_mc_queue *)

procedure   execute_processing_on_general_mc_starts;

var     assly_no,
        lot_no,
        queue_lg,
        queue_lg_no,
        size,
        mc_no_to_visit    : integer;

        process_finish_time : real;

begin (* execute_processing_on_general_mc_starts *)

    assly_no:=ptr_to_current_executable_event^.array_of_relevant_nos[1];
    lot_no:=ptr_to_current_executable_event^.array_of_relevant_nos[2];
    mc_no_to_visit:=ptr_to_current_executable_event^.
                        array_of_relevant_nos[3];
    mc_busy[mc_no_to_visit]:= true;
    size:=ala[assly_no].
             dol[lot_no].
                original_size;
    process_finish_time:=
    ptr_to_current_executable_event^.execution_time +
    ala[assly_no].
      dol[lot_no].cpt;

(* now insert an event called processing gets over on the same machine *)

    newmodified(new_event);
    new_event^.event:=processing_on_general_mc_gets_over;
    new_event^.array_of_relevant_nos[1]:=assly_no;
    new_event^.array_of_relevant_nos[2]:=lot_no;
    new_event^.array_of_relevant_nos[3]:=mc_no_to_visit;
    new_event^.execution_time:=process_finish_time;
    plug_in_the_event(new_event);
```

```
(*  the event has been inserted *)

    queue_lg:=queue_before_general_mc[mc_no_to_visit].queue_length;
    for queue_lg_no:=2 to queue_lg do
    queue_before_general_mc[mc_no_to_visit].
                    details_of_queue[queue_lg_no-1] :=
    queue_before_general_mc[mc_no_to_visit].
                              details_of_queue[queue_lg_no];
    queue_before_general_mc[mc_no_to_visit].queue_length:=
    queue_before_general_mc[mc_no_to_visit].queue_length - 1;

(*  now we modify the assly-no and lot-no data *)
   ala[assly_no].dol[lot_no].time_when_processing_starts:=
            ptr_to_current_executable_event^.execution_time;
   with ala[assly_no].
          dol[lot_no] do
   begin (* with *)
     current_mc_no:=mc_no_to_visit;
     total_wait_time_in_queues_so_far:=
     total_wait_time_in_queues_so_far +
 {    ptr_to_current_executable_event^.execution_time -}
     time_when_processing_starts -
                time_of_joining_the_queue;
   end; (* with *)

end; (* execute_processing_on_general_mc_starts *)

procedure   execute_processing_on_assly_mc_starts;

var    assly_no,
       lot_no,
       queue_lg,
       size,
       lot_size,
       last_quantity_available,
       quantity_available,
       quantity_to_be_contributed,
       children_index,
       no_of_children,
       lowlim,
       lowlim_index,
       lot_no_of_child,
       assly_no_of_child,
       qty_picked_up,
       mc_no_to_visit    : integer;

       process_finish_time : real;

begin (* execute_processing_on_assly_mc_starts *)

    assly_no:=ptr_to_current_executable_event^.array_of_relevant_nos[1];
    lot_no:=ptr_to_current_executable_event^.array_of_relevant_nos[2];
    mc_no_to_visit:=ptr_to_current_executable_event^.
                        array_of_relevant_nos[3];
    lot_size:=
    ala[assly_no].
        dol[lot_no].original_size;
```

```
no_of_children:=abda[assly_no].
                      no_of_children_for_assembly;
for children_index:=1 to no_of_children do
begin (* for children_index *)
  quantity_to_be_contributed:=
  abda[assly_no].
     array_of_child_assly[children_index].qty_to_be_contributed
     * lot_size;
  lowlim:=1;
  lot_no_of_child:=
  queue_before_assly_mc[mc_no_to_visit].
      details_of_queues[children_index].
           details_of_queue[lowlim].
               lot_no;
  assly_no_of_child:=
  queue_before_assly_mc[mc_no_to_visit].
      details_of_queues[children_index].
          details_of_queue[lowlim].
               assly_no;
  last_quantity_available:=0;
  quantity_available:=
  ala[assly_no_of_child].
      dol[lot_no_of_child].qty_left;
  while (quantity_available < quantity_to_be_contributed) do
  begin (* while *)
    qty_picked_up:=quantity_available-last_quantity_available;
    ala[assly_no_of_child].
         dol[lot_no_of_child].
             total_wait_time_in_queues_so_far:=
    ala[assly_no_of_child].
         dol[lot_no_of_child].
             total_wait_time_in_queues_so_far +
    ((ptr_to_current_executable_event^.execution_time -
      ala[assly_no_of_child].
           dol[lot_no_of_child].
              time_of_entry_into_the_shop)*qty_picked_up/
    ala[assly_no_of_child].
         dol[lot_no_of_child].original_size);
    lowlim:=lowlim + 1;
    lot_no_of_child:=
    queue_before_assly_mc[mc_no_to_visit].
        details_of_queues[children_index].
           details_of_queue[lowlim].
               lot_no;
    assly_no_of_child:=
       queue_before_assly_mc[mc_no_to_visit].
          details_of_queues[children_index].
              details_of_queue[lowlim].
                 assly_no;
    last_quantity_available:=quantity_available;
    quantity_available:=
    quantity_available +
    ala[assly_no_of_child].
        dol[lot_no_of_child].qty_left;
  end; (* while *)
  if (quantity_available = quantity_to_be_contributed)
  then
  else  (* quantity_available > quantity_to_be_contributed *)
```

```
      begin
        qty_picked_up:=quantity_available-last_quantity_available;
        ala[assly_no_of_child].
           dol[lot_no_of_child].qty_left:=
        quantity_available - quantity_to_be_contributed;
        ala[assly_no_of_child].
          dol[lot_no_of_child].
              total_wait_time_in_queues_so_far:=
        ala[assly_no_of_child].
          dol[lot_no_of_child].
              total_wait_time_in_queues_so_far +
        ((ptr_to_current_executable_event^.execution_time -
          ala[assly_no_of_child].
             dol[lot_no_of_child].
                time_of_entry_into_the_shop)*qty_picked_up/
        ala[assly_no_of_child].
          dol[lot_no_of_child].original_size);
        lowlim:=lowlim - 1;
      end;

    (* lowlim represents now the nos. in queue which get absorbed
       in the assly *)

    queue_lg:=
    queue_before_assly_mc[mc_no_to_visit].
       details_of_queues[children_index].
          queue_length;
    for lowlim_index:=(lowlim+1) to  queue_lg do
    queue_before_assly_mc[mc_no_to_visit].
       details_of_queues[children_index].
          details_of_queue[lowlim_index-lowlim]:=
    queue_before_assly_mc[mc_no_to_visit].
       details_of_queues[children_index].
          details_of_queue[lowlim_index];
    if (queue_lg >= lowlim) then
    queue_before_assly_mc[mc_no_to_visit].
       details_of_queues[children_index].
          queue_length:=queue_lg - lowlim
    else
    queue_before_assly_mc[mc_no_to_visit].
       details_of_queues[children_index].
          queue_length:=0;

    queue_before_assly_mc[mc_no_to_visit].
       quantity_available[children_index]:=
       quantity_available - quantity_to_be_contributed;
  end; (* for children_index *)

  process_finish_time:=
  ptr_to_current_executable_event^.execution_time +
  lot_size *
  aam[mc_no_to_visit].processing_time_per_unit;
  newmodified(new_event);
  new_event^.event:=processing_on_assly_mc_gets_over;
  new_event^.array_of_relevant_nos[1]:=assly_no;
  new_event^.array_of_relevant_nos[2]:=lot_no;
  new_event^.array_of_relevant_nos[3]:=mc_no_to_visit;
  new_event^.execution_time:=process_finish_time;
```

```
    plug_in_the_event(new_event);

end; (* execute_processing_on_assly_mc_starts *)

procedure   execute_processing_on_general_mc_gets_over;

var    assly_no,
       lot_no,
       queue_lg,
       queue_lg_no,
       size,
       next_process_no,
       mc_no_to_visit    : integer;

begin (* execute_processing_on_general_mc_gets_over *)

    assly_no:=ptr_to_current_executable_event^.array_of_relevant_nos[1];
    lot_no:=ptr_to_current_executable_event^.array_of_relevant_nos[2];
    mc_no_to_visit:=ptr_to_current_executable_event^.
                        array_of_relevant_nos[3];
    mc_busy[mc_no_to_visit]:=false;
    if(queue_before_general_mc[mc_no_to_visit].queue_length > 0) then
         begin
           newmodified(new_event);
           new_event^.event:=processing_on_general_mc_starts;
           new_event^.array_of_relevant_nos[1]:=
                   queue_before_general_mc[mc_no_to_visit].
                                details_of_queue[1].assly_no;
           new_event^.array_of_relevant_nos[2]:=
                   queue_before_general_mc[mc_no_to_visit].
                                details_of_queue[1].lot_no;
           new_event^.array_of_relevant_nos[3]:=mc_no_to_visit;
           new_event^.execution_time:=
           ptr_to_current_executable_event^.execution_time;
           plug_in_the_event(new_event);
         end;

(* beginning of initialisations and updations *)

    ala[assly_no].
      dol[lot_no].no_of_agm_visited
          :=
    ala[assly_no].
      dol[lot_no].no_of_agm_visited + 1;
    ala[assly_no].
      dol[lot_no].tpt_completed_so_far:=
    ala[assly_no].
      dol[lot_no].tpt_completed_so_far +
    ala[assly_no].
      dol[lot_no].cpt;
    agm[mc_no_to_visit].tpt:=
    agm[mc_no_to_visit].tpt +
                          ala[assly_no].
                       dol[lot_no].cpt;

    ala[assly_no].
      dol[lot_no].priority_value:=
    (ala[assly_no].
```

```
      dol[lot_no].total_required_processing_time -
    ala[assly_no].
      dol[lot_no].tpt_completed_so_far -
   (clock_time -
   ala[assly_no].
     dol[lot_no].required_completion_time) )/
   ala[assly_no].
      dol[lot_no].total_required_processing_time;

(* end of initialisations and updations *)

   if (ala[assly_no].
          dol[lot_no].
              no_of_agm_visited) =
      (abda[assly_no].
          no_of_processes_needed)
   then (* all the processes are done *)
   begin (* then *)
     ala[assly_no].
         dol[lot_no].
           processing_complete:=true;
     ala[assly_no].
         dol[lot_no].actual_completion_time:=
     ptr_to_current_executable_event^.execution_time;
     assly_no:=
     abda[assly_no].
        assly_no_after_processing;
     ala[assly_no].
         dol[lot_no].time_of_entry_into_the_shop:=
     ptr_to_current_executable_event^.execution_time;
     if (abda[assly_no].parent_no = 0)
     then (* joins the finished goods queue *)
       begin
         newmodified(new_event);
         new_event^.event:=lot_enters_the_finished_product_list;
         new_event^.array_of_relevant_nos[1]:=assly_no;
         new_event^.array_of_relevant_nos[2]:=lot_no;
         new_event^.array_of_relevant_nos[3]:=0;
         new_event^.execution_time:=
         ptr_to_current_executable_event^.execution_time;
         plug_in_the_event(new_event);
       end
     else (* joins the assly machine queue *)
       begin
         mc_no_to_visit:=
         abda[assly_no].assly_on_mc_no;
         newmodified(new_event);
         new_event^.event:=lot_enters_the_assly_mc_queue;
         new_event^.array_of_relevant_nos[1]:=assly_no;
         new_event^.array_of_relevant_nos[2]:=lot_no;
         new_event^.array_of_relevant_nos[3]:=mc_no_to_visit;
         new_event^.execution_time:=
         ptr_to_current_executable_event^.execution_time;
         plug_in_the_event(new_event);
       end;
   end  (* then *)

   else (* joins the next machine *)
```

```
    if (ala[assly_no].
          dol[lot_no].
              no_of_agm_visited) <
       (abda[assly_no].
          no_of_processes_needed)
    then
    begin (* else *)
      next_process_no:=
      ala[assly_no].
           dol[lot_no].
               no_of_agm_visited+1;
      mc_no_to_visit:=
      abda[assly_no].
          aps[next_process_no].mc_no;
      newmodified(new_event);
      new_event^.event:=lot_enters_the_general_mc_queue;
      new_event^.array_of_relevant_nos[1]:=assly_no;
      new_event^.array_of_relevant_nos[2]:=lot_no;
      new_event^.array_of_relevant_nos[3]:=mc_no_to_visit;
      new_event^.execution_time:=
      ptr_to_current_executable_event^.execution_time;
      plug_in_the_event(new_event);
    end; (* else *)

end; (* execute_processing_on_general_mc_gets_over *)

procedure   execute_processing_on_assly_mc_gets_over;

var    assly_no,
       finished_assly_no,
       lot_no,
       next_process_no,
       mc_no_to_visit    : integer;

begin (* execute_processing_on_assly_mc_gets_over *)

    assly_no:=ptr_to_current_executable_event^.array_of_relevant_nos[1];
    lot_no:=ptr_to_current_executable_event^.array_of_relevant_nos[2];
    mc_no_to_visit:=ptr_to_current_executable_event^.
                        array_of_relevant_nos[3];
    ala[assly_no].
            lot_no_which_will_enter_the_shop:=
    ala[assly_no].
            lot_no_which_will_enter_the_shop + 1;
    ala[assly_no].
          dol[lot_no].time_of_entry_into_the_shop:=
    ptr_to_current_executable_event^.execution_time;
    ala[assly_no].dol[lot_no].actual_completion_time:=
        ptr_to_current_executable_event^.execution_time;

    if (abda[assly_no].
                 immediately_gets_into_assly)
    then
      begin (* gets into assly *)
        finished_assly_no:=
        abda[assly_no].parent_no;
        mc_no_to_visit:=
        abda[finished_assly_no].assly_on_mc_no;
```

```
        newmodified(new_event);
        new_event^.event:=lot_enters_the_assly_mc_queue;
        new_event^.array_of_relevant_nos[1]:=assly_no;
        new_event^.array_of_relevant_nos[2]:=lot_no;
        new_event^.array_of_relevant_nos[3]:=mc_no_to_visit;
        new_event^.execution_time:=
        ptr_to_current_executable_event^.execution_time;
        plug_in_the_event(new_event);
      end (* gets into assly *)
    else if (abda[assly_no].
                immediately_needs_processing)
         then
          begin (* processing of the assembly starts *)
            mc_no_to_visit:=
            abda[assly_no].
                    aps[1].mc_no;
            newmodified(new_event);
            new_event^.event:=lot_enters_the_general_mc_queue;
            new_event^.array_of_relevant_nos[1]:=assly_no;
            new_event^.array_of_relevant_nos[2]:=lot_no;
            new_event^.array_of_relevant_nos[3]:=mc_no_to_visit;
            new_event^.execution_time:=
            ptr_to_current_executable_event^.execution_time;
            plug_in_the_event(new_event);
          end (* processing of the assembly starts *)
         else
          begin (* gets into the finished product list *)
            newmodified(new_event);
            new_event^.event:=lot_enters_the_finished_product_list;
            new_event^.array_of_relevant_nos[1]:=assly_no;
            new_event^.array_of_relevant_nos[2]:=lot_no;
            new_event^.array_of_relevant_nos[3]:=0;
            new_event^.execution_time:=
            ptr_to_current_executable_event^.execution_time;
            plug_in_the_event(new_event);
          end; (* gets into the finished product list *)

end; (* execute_processing_on_assly_mc_gets_over *)

procedure      execute_the_event;

begin (* execute_the_event *)

    case ptr_to_current_executable_event^.event of

      collect_statistics_from_now_on                  :
          execute_collect_statistics_from_now_on;
      lot_enters_the_shop                  :
          execute_lot_enters_the_shop;
      lot_enters_the_general_mc_queue      :
          execute_lot_enters_the_general_mc_queue;
      lot_enters_the_assly_mc_queue        :
          execute_lot_enters_the_assly_mc_queue;
      processing_on_general_mc_starts      :
           execute_processing_on_general_mc_starts;
      processing_on_assly_mc_starts        :
          execute_processing_on_assly_mc_starts;
      processing_on_general_mc_gets_over   :
```

```
            execute_processing_on_general_mc_gets_over;
        processing_on_assly_mc_gets_over     :
            execute_processing_on_assly_mc_gets_over;
        lot_enters_the_finished_product_list :
            execute_lot_enters_the_finished_product_list;
    end; (* case *)
    event_addfreelist(ptr_to_current_executable_event);

end; (* execute_the_event *)

procedure      collect_and_print_statistics;

var    assly_no,
       lot_no,
       no_of_orders_placed,
       order_no,
       current_demand_qty,
       current_demand_period,
       current_supply_qty,
       current_supply_period,
       machine_no,
       queue_length,
       queue_length_index,
       no_of_queues,
       queue_no,
       no_of_lots ,
       general_mc,
       qty,period ,
       mc_no       : integer;
       setup_cost,
       total_setup_cost,
       setup_cost_reductions,
       total_aggregate_lateness,
       inventory_carrying_cost,
       total_theoretical_inventory_carrying_cost,
       theoretical_inventory_carrying_cost,
       total_inventory_carrying_cost : real;
       sum_util,
       av_util,
       sum_range,
       coordination_range    :real;
       range:array[range_of_assly_mcs] of real;

begin (* collect_and_print_statistics *)

(* calculating theoretical inventory carrying costs *)

    theoretical_inventory_carrying_cost:=0.0;
    for assly_no:=1 to max_assly_no do
    begin (* for assly_no *)
     (* theoretical_inventory_carrying_cost:=
      theoretical_inventory_carrying_cost +
      array_for_output_of_w_w[assly_no].holding_cost;
      theoretical_cost[assly_no]:=
      array_for_output_of_w_w[assly_no].holding_cost;*)
      qty:=0;
      period:=1;
      for order_no:= 1 to array_for_output_of_w_w[assly_no].
```

```
                                        no_of_orders_placed do
    begin
        qty:=qty+array_for_output_of_w_w[assly_no].
                            order_schedule[order_no].qty_produced;
        while (qty-demand[assly_no,period]>0) and
                                    (period<max_period_no) do
        begin
            theoretical_cost[assly_no]:=
                  (qty-demand[assly_no,period]) *
                     abda[assly_no].cost*inventory_cost_fraction
                                          + theoretical_cost[assly_no];
            theoretical_inventory_carrying_cost:=
               theoretical_inventory_carrying_cost +
                  (qty-demand[assly_no,period]) *
                     abda[assly_no].cost*inventory_cost_fraction;
            period:=period+1;
            qty:=qty-demand[assly_no,period];
         end;(* while qty >*)
      end;(* for- order_no *)
     no_of_lots:=ala[assly_no].no_of_lots;
     for lot_no:=1 to no_of_lots do
     begin (* for lot_no *)
       actual_cost[assly_no]:=actual_cost[assly_no]+
       ala[assly_no].
          dol[lot_no].total_wait_time_in_queues_so_far *
          abda[assly_no].cost * inventory_cost_fraction ;
     end;(* for lot no *)
    end; (* for assly_no *)

(* end of calculating theoretical inventory carrying costs *)

(* adding waiting times for asslys in queues also *)

   for machine_no:=1 to max_mc_no do
   begin (* for general_machine_no *)
     queue_length:=queue_before_general_mc[machine_no].queue_length;
     for queue_length_index:=1 to queue_length do
     begin (* for queue_no *)
       assly_no:=queue_before_general_mc[machine_no].
                       details_of_queue[queue_length_index].assly_no;
      lot_no:=queue_before_general_mc[machine_no].
                       details_of_queue[queue_length_index].lot_no;
      ala[assly_no].
        dol[lot_no].total_wait_time_in_queues_so_far:=
      ala[assly_no].
        dol[lot_no].total_wait_time_in_queues_so_far +
      simulation_time -
      ala[assly_no].
        dol[lot_no].time_of_joining_the_queue;
     end; (* for queue_no *)
   end; (* for general_machine_no *)
   for machine_no:=16 to 20 do
   begin (* for assly_machine_no *)
     no_of_queues:=queue_before_assly_mc[machine_no].
                             no_of_subasslys_into_assly;
     for queue_no:=1 to no_of_queues do
     begin (* for queue_no *)
       queue_length:=queue_before_assly_mc[machine_no].
```

```
                    details_of_queues[queue_no].queue_length;
      for queue_length_index:=1 to queue_length do
      begin (* for queue_length_index *)
         assly_no:=
         queue_before_assly_mc[machine_no].
            details_of_queues[queue_no].
                details_of_queue[queue_length_index].assly_no;
         lot_no:=
         queue_before_assly_mc[machine_no].
            details_of_queues[queue_no].
                details_of_queue[queue_length_index].lot_no;
         ala[assly_no].
             dol[lot_no].total_wait_time_in_queues_so_far:=
         ala[assly_no].
             dol[lot_no].total_wait_time_in_queues_so_far +
         simulation_time -
         ala[assly_no].
         dol[lot_no].time_of_joining_the_queue;
      end; (* for queue_length_index *)
    end; (* for queue_no *)
  end; (* for assly_machine_no *)

(* end of adding waiting times for asslys in queues also *)

(* calculating actual inventory carrying costs *)

  total_inventory_carrying_cost:=0.0;
  for assly_no:=1 to max_assly_no do
  begin (* for assly_no *)
    no_of_lots:=ala[assly_no].no_of_lots;
    for lot_no:=1 to no_of_lots do
    begin (* for lot_no *)
      inventory_carrying_cost:=
      ala[assly_no].
         dol[lot_no].total_wait_time_in_queues_so_far *
         abda[assly_no].cost * inventory_cost_fraction/
                                 no_of_hours_in_a_period;
      { inventory_carrying_cost/no_of_hours_in_a_year; }
      ala[assly_no].
         dol[lot_no].actual_inventory_carrying_cost:=
      inventory_carrying_cost;
      total_inventory_carrying_cost:=
      total_inventory_carrying_cost + inventory_carrying_cost;
      abda[assly_no].
        actual_inventory_carrying_cost:=
      abda[assly_no].
        actual_inventory_carrying_cost +
      inventory_carrying_cost;
    end; (* for lot_no *)
  end; (* for assly_no *)

(* end of calculating actual inventory carrying costs *)

(* calculating setup costs *)
  total_setup_cost:=0.0;
  setup_cost_reductions:=0.0;
  for assly_no:=1 to max_assly_no do
  begin (* for *)
```

```
    setup_cost:=array_for_output_of_w_w[assly_no].order_cost;
    total_setup_cost:=total_setup_cost + setup_cost;
    no_of_orders_placed:=
    array_for_output_of_w_w[assly_no].no_of_orders_placed;
 for order_no:=1 to  no_of_orders_placed do
    begin (* for order_no *)
      if (array_for_output_of_w_w[assly_no].
                 order_schedule[order_no].period_no *
                    no_of_hours_in_a_period <
          collect_statistics_from_time)
      then setup_cost_reductions:=
           setup_cost_reductions +
           abda[assly_no].total_setup_cost
      else if (array_for_output_of_w_w[assly_no].
                   order_schedule[order_no].period_no *
                      no_of_hours_in_a_period >
                simulation_time)
           then setup_cost_reductions:=
                setup_cost_reductions +
                abda[assly_no].
                        total_setup_cost;
    end; (* for order_no *)
  end; (* for *)
  total_setup_cost:=total_setup_cost - setup_cost_reductions;

(* end of calculating setup costs *)

(* calculating aggregate_lateness *)

  total_aggregate_lateness:=0.0;
  for assly_no:=1 to max_assly_no do
  begin (* for assly_no *)
    no_of_lots:=ala[assly_no].no_of_lots;
    for lot_no:=1 to no_of_lots do
    begin (* for lot_no *)
      if (assly_no = 1) then
      ala[1].dol[lot_no].actual_completion_time:= clock_time;
      ala[assly_no].
         dol[lot_no].aggregate_lateness_of_lot:=
      (ala[assly_no].
         dol[lot_no].actual_completion_time -
       ala[assly_no].
         dol[lot_no].required_completion_time) *
       ala[assly_no].
         dol[lot_no].original_size;
       total_aggregate_lateness:=
       total_aggregate_lateness +
       ala[assly_no].
         dol[lot_no].aggregate_lateness_of_lot;
    end; (* for lot_no *)
  end; (* for assly_no *)

(* end of calculating aggregate lateness *)


(* calculating units_lateness *)

  total_units_lateness:=0.0;
```

```
  for assly_no:=1 to max_assly_no do
  begin (* for assly_no *)
    no_of_lots:=ala[assly_no].no_of_lots;
    current_demand_qty:=0;
    current_demand_period:=0;
    for lot_no:=1 to no_of_lots do
    begin (* for lot_no *)
      current_supply_qty:=
      ala[assly_no].
         dol[lot_no].original_size;
      current_supply_period:=
      trunc(ala[assly_no].
              dol[lot_no].actual_completion_time/
                 no_of_hours_in_a_period) + 1;
      while (current_supply_qty > 0)
                        and
             (current_demand_period < max_period_no)
                        do
      begin (* while current_supply_qty > 0 *)
           current_demand_period:=current_demand_period + 1;
           current_demand_qty:=demand[assly_no,current_demand_period];
         if (current_supply_period > current_demand_period)
         then
         begin (* then *)
           ala[assly_no].
              dol[lot_no].units_lateness_of_lot:=
           (current_supply_period - current_demand_period) *
           current_demand_qty;
           total_units_lateness:=
           total_units_lateness +
           (current_supply_period - current_demand_period) *
           current_demand_qty;
         end; (* then *)
         current_supply_qty:=current_supply_qty-current_demand_qty;
      end; (* while current_supply_qty > 0 *)
    end; (* for lot_no *)
  end; (* for assly_no *)

(* end of calculating units_lateness *)

(* calculating mc utilisation statistics *)

   for machine_no:=1 to max_mc_no do
   begin (* for machine_no *)
     agm[machine_no].cu:=
     agm[machine_no].tpt/
     (clock_time - collect_statistics_from_time);
   end; (* for machine_no *)

   for machine_no:=16 to 20 do
   begin (* for machine_no *)
     aam[machine_no].cu:=
     aam[machine_no].tpt/
     (clock_time - collect_statistics_from_time);
   end; (* for machine_no *)

(* end of calculating mc utilisation statistics *)
```

```
  if (total_units_lateness < best_sol) then
  begin
      rewrite(potty_output,'s.res');
      writeln(potty_output,'total aggregate lateness =',
                                              total_aggregate_lateness);
      writeln(potty_output,'total setup cost =',total_setup_cost);
      writeln(potty_output,'total units lateness =',total_units_lateness);
      writeln(potty_output,'total inventory carrying cost =',
                                        total_inventory_carrying_cost);

      writeln(potty_output,'theoretical inventory carrying cost =',
                                     theoretical_inventory_carrying_cost);

      sum_util:=0.0;
      for general_mc:= 1 to max_no_of_proces_on_any_assly do
      sum_util:=sum_util+agm[general_mc].cu;
      av_util:=sum_util/max_no_of_proces_on_any_assly;
      for general_mc:= 1 to max_no_of_proces_on_any_assly do
      writeln(potty_output,
           'Utilisation for general machine',general_mc:2,' =',
                                              agm[general_mc].cu:5);

      writeln(potty_output,'Average Utilisation is',' =',av_util:5);
      sum_range:=0;
      for mc_no:= 16 to 20 do

      begin
          range[mc_no]:=latest_lot_time[mc_no]-
                              earliest_lot_time[mc_no];
          writeln(potty_output,'coordination_range for assly centre',mc_no:2,
                        'is',range[mc_no]);
          sum_range:=sum_range+range[mc_no];
      end;
      coordination_range:=sum_range/(20-15);
      if first_run then
      prev_range:=coordination_range;

      writeln(potty_output,'previous coordination range is',' =',
                                    prev_range  :5);
      writeln(potty_output,'improved coordination range is',' =',

                                    coordination_range  :5);
      close(potty_output);
      rewrite(s_dat,'s.dat');
      writeln(s_dat,total_aggregate_lateness);
      writeln(s_dat,total_setup_cost);
      writeln(s_dat,total_units_lateness);
      writeln(s_dat,total_inventory_carrying_cost);
      writeln(s_dat,theoretical_inventory_carrying_cost);
      writeln(s_dat,av_util);
      writeln(s_dat,coordination_range  );
      close(s_dat);
      for general_mc:= 1 to max_no_of_proces_on_any_assly do
      writeln('Utilisation for general machine',general_mc:2,' =',
                                              agm[general_mc].cu:5);
      writeln('Average Utilisation is',' =',av_util:5);
  end;
end; (* collect_and_print_statistics *)
```

```
procedure set_parameters;
var
  assly,
  assly_no,
  lot_no,
  no_of_lots,
  mc_no         :integer;

begin
    for assly_no:= 1 to max_assly_no do
    if abda[assly_no].immediately_needs_processing then
    begin
         assly:=abda[assly_no].assly_no_after_processing;
         critical_child[assly]:=assly_no;
         no_of_lots:=array_for_output_of_w_w[assly_no].
                                      no_of_orders_placed;
         for lot_no:= 1 to no_of_lots do
         begin
             time_of_reaching[assly_no,lot_no]:=0;
         end;
             coordination_index[assly_no]:=1;
    for mc_no:= 16 to 20 do
    latest_assly_time[assly_no,mc_no]:=0;
    end;
    for mc_no:= 16 to 20 do
    begin
        latest_lot_time[mc_no]:=0;
        earliest_lot_time[mc_no]:=0;
        max_time[mc_no]:=0;
        assly_count[mc_no]:=0;
        present_mean_time[mc_no]:=0;
    end;
end;

procedure next_set_parameters;
var
  assly,
  assly_no,
  lot_no,
  no_of_lots,
  mc_no         :integer;

begin
    for assly_no:= 1 to max_assly_no do
    if abda[assly_no].immediately_needs_processing then
    begin
         no_of_lots:=array_for_output_of_w_w[assly_no].
                                      no_of_orders_placed;

         for lot_no:= 1 to no_of_lots do
         begin
             time_of_reaching[assly_no,lot_no]:=0;
         end;
    for mc_no:= 16 to 20 do
    latest_assly_time[assly_no,mc_no]:=0;
    end;
```

```
    for mc_no:= 16 to 20 do
    begin
        latest_lot_time[mc_no]:=0;
        earliest_lot_time[mc_no]:=0;
        max_time[mc_no]:=0;
        prev_mean_time[mc_no]:=present_mean_time[mc_no];
        present_mean_time[mc_no]:=0;
    end;
end;


procedure run;
var
  a,b:integer;
begin
    read_input_data;
    do_explosion;
    do_lot_sizing;
    initialisations;
    best_sol:=infinity;
    set_parameters;
    prepare_queue_of_lots;
    prepare_events_regarding_entry_of_components;
    prepare_collect_statistic_event;

(*    decide_times_of_entry_in_the_shop;    *)
    open_print_the_event_list;
    while (clock_time < simulation_time) and (flag = false) do
    begin
      find_the_next_executable_event;
      if (flag=false) then
      begin
        print_the_event_list;
        execute_the_event;
      end;
    end;
    close_print_the_event_list;
    collect_and_print_statistics;
end;


procedure next_run;
var
  a,b:integer;
begin
    do_lot_sizing;
    initialisations;
    next_set_parameters;
    prepare_queue_of_lots;
    prepare_events_regarding_entry_of_components;
    prepare_collect_statistic_event;

    open_print_the_event_list;
    while (clock_time < simulation_time) and (flag = false) do
    begin
      find_the_next_executable_event;
      if (flag=false) then
      begin
```

```
        print_the_event_list;
        execute_the_event;
      end;
    end;
    close_print_the_event_list;
    collect_and_print_statistics;
end;



begin (* main *)
  times:=0;
  first_run:=true;
  run;
  first_run:=false;
  (* while (total_units_lateness <= best_sol) and (times < 10) do*)
  while (times<20) do
  begin
      times:=times+1;
      writeln(times,'iteration');
      if (total_units_lateness<best_sol) then
      begin
          writeln('improved');
          best_sol:=total_units_lateness;
      end;
      next_run;
  end;
end.
```

APPENDIX B
DETAILS OF TWENTY TEST PROBLEMS AND RESULTS
(INCLUDING t TEST RESULTS)

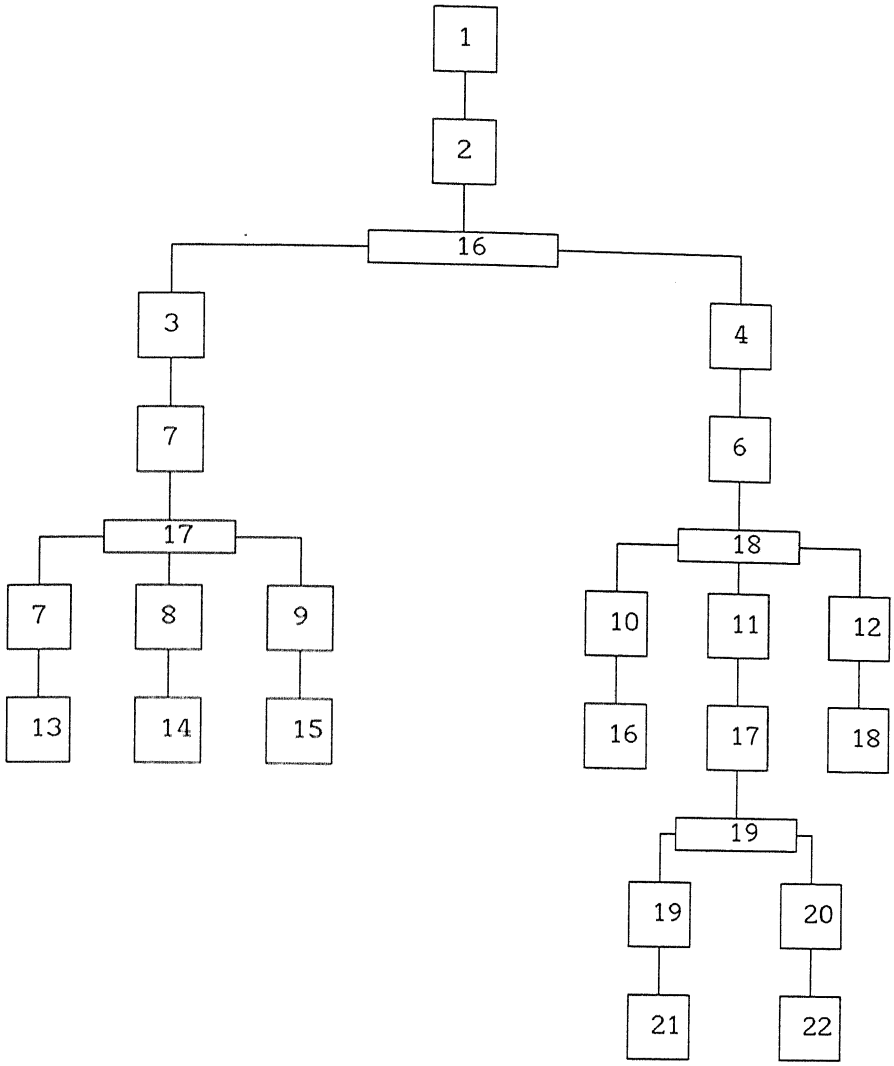


FIGURE B.1 PROBLEM 1

Component No.	Routing Sequence
2	3 → 4
5	1 → 2 → 3
6	3 → 4 → 1
13	4 → 1 → 2 → 3
14	3 → 2 → 4
15	4 → 1 → 2
16	3 → 4 → 1
17	4 → 1 → 2
18	2 → 1 → 3
21	1 → 2 → 4 → 3
22	3 → 2 → 4

TABLE B.1 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
1	16	1,4,7,12,15,17,20,22,26,30,35,39,42,46,50

TABLE B.2 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	705149	748046	692481	671361
TOTAL SETUP COST	1645	1721	1645	1645
TOTAL UNITS LATENESS	756	712	756	701
TOTAL INVENTORY CARRYING COST	24400400	27400500	24400400	22600400
THEORETICAL INVENTORY CARRYING COST	11107.7	10528	11107.7	11107.7
AVERAGE UTILISATION	0.7443	0.757817	0.7443	0.774761
AVERAGE COORDINATION RANGE	713.5	371.25	713.5	283.6

TABLE B.3 RESULTS (PROBLEM 1)

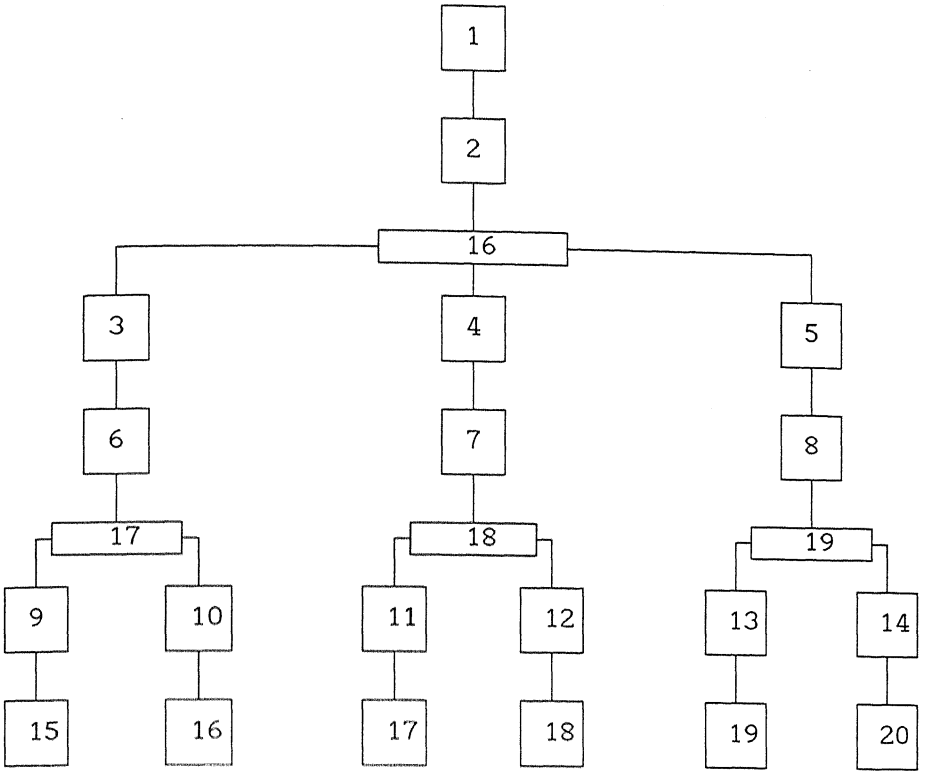


FIGURE B.2 PROBLEM 2

Component No.	Routing Sequence
2	2 → 4
6	1 → 3
7	4 → 2
8	3 → 4
15	4 → 1
16	3 → 2
17	1 → 2
18	2 → 4
19	4 → 3
20	3 → 1

TABLE B.4 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
1	9	1,4,7,19,29,35,41,47,50,
3	5	7,14,21,35,42
4	5	6,18,30,36,48
5	5	9,18,27,36,45

TABLE B.5 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	778019	791597	782741	750593
TOTAL SETUP COST	740	780	740	740
TOTAL UNITS LATENESS	426	345	394	312
TOTAL INVENTORY CARRYING COST	23800000	26400000	23800000	23400000
THEORETICAL INVENTORY CARRYING COST	11434.3	11434.3	11434.3	11434.3
AVERAGE UTILISATION	0.673021	0.780576	0.692023	0.731424
AVERAGE COORDINATION RANGE	479	312	279	184

TABLE B.6 RESULTS (PROBLEM 2)

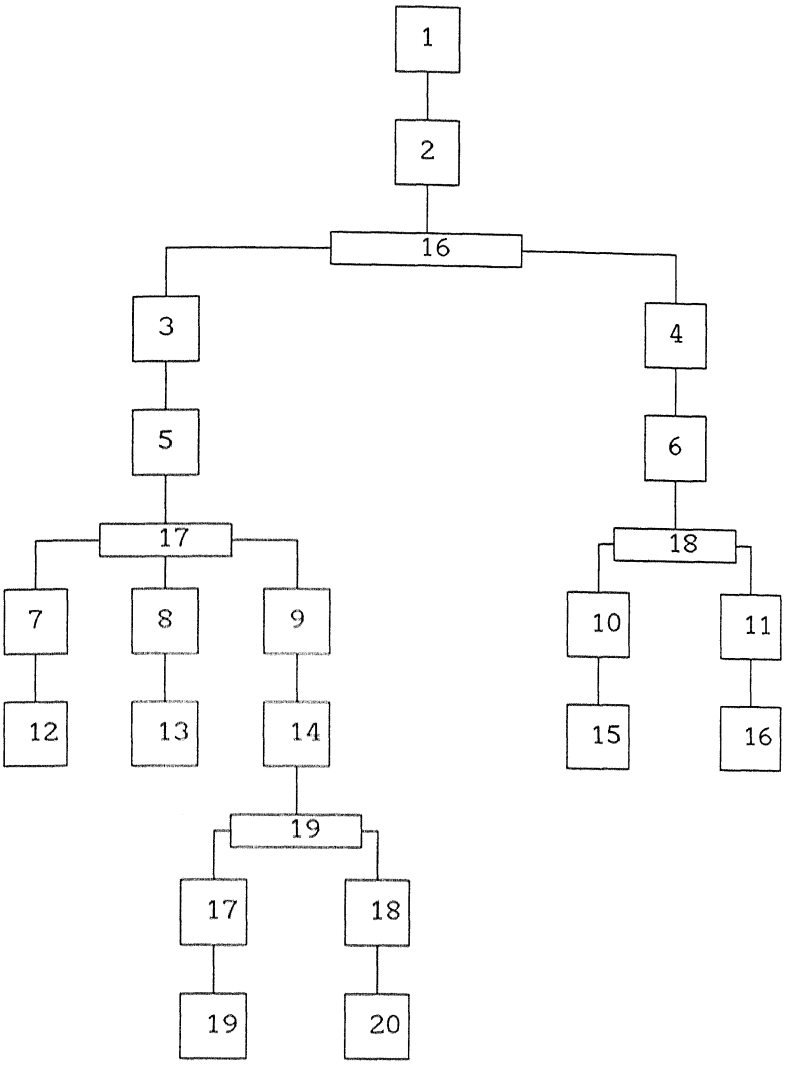


FIGURE B.3 PROBLEM 3

Component No.	Routing Sequence
2	1 → 2
5	3 → 4 → 1
6	1 → 2 → 4
12	2 → 1
13	4 → 3
14	1 → 2
15	3 → 2
16	4 → 1
19	1 → 3
20	2 → 1

TABLE B.7 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
1	12	1,3,7,13,17,27,31,33,37,41,48,50
3	4	6,16,36,46
4	5	7,13,20,27,37
9	5	2,11,20,29,39

TABLE B.8 DEMAND DETAILS

TOTAL AGGREGATE LATENESS	995432	995540	1020560	1021470
TOTAL SETUP COST	1265	1265	1265	1265
TOTAL UNITS LATENESS	802	802	744	698
TOTAL INVENTORY CARRYING COST	44199900	44199900	44199900	41999900
THEORETICAL INVENTORY CARRYING COST	15393	15393	15393	15393
AVERAGE UTILISATION	0.713262	0.713262	0.738163	0.816929
AVERAGE COORDINATION RANGE	746.25	453.5	746.25	307.6

TABLE B.9 RESULTS (PROBLEM 3)

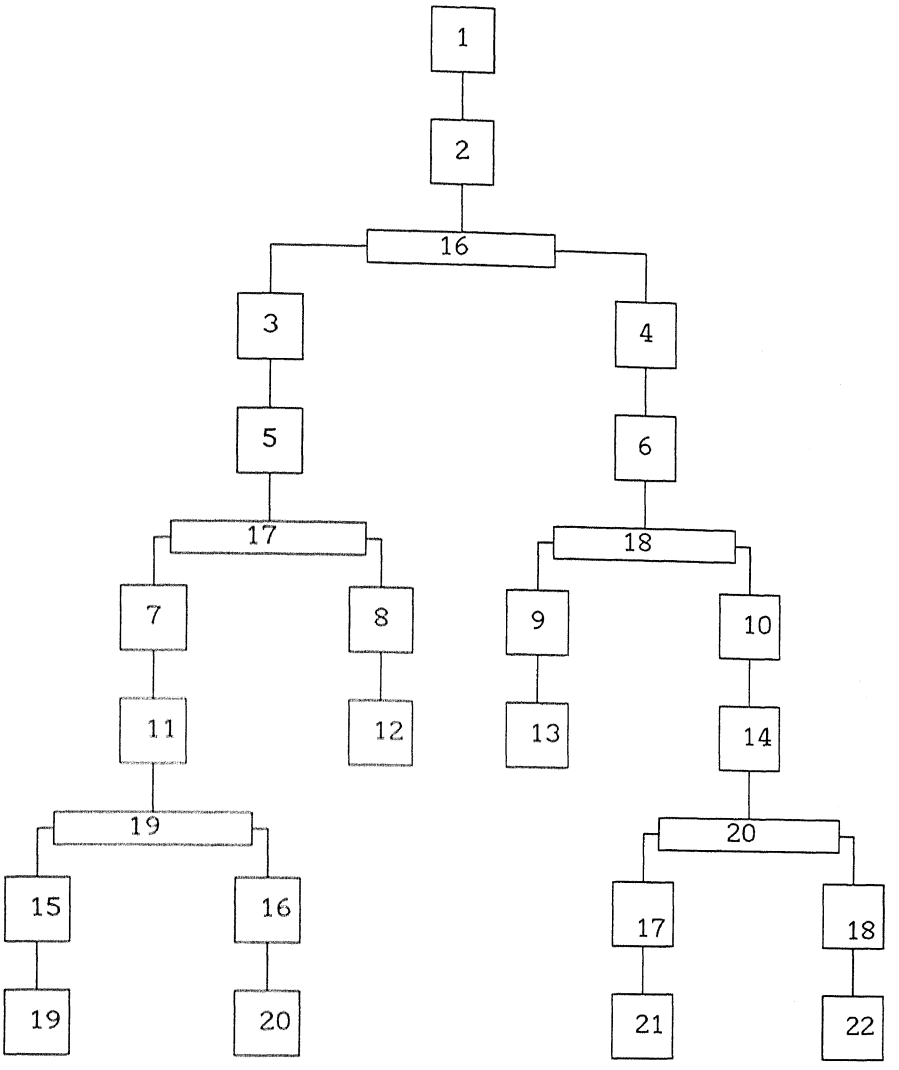


FIGURE B.4 PROBLEM 4

Component No.	Routing Sequence
2	1 → 3 → 5
5	2 → 4 → 5
6	1 → 3 → 5
11	1 → 2
12	3 → 4
13	5 → 1
14	2 → 3
19	1 → 2 → 3
20	3 → 4 → 5
21	2 → 3 → 4
22	4 → 5 → 1

TABLE B.10 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
1	16	1,4,7,10,12,15,17,20,22,26,30,35,39,42,46,50

TABLE B.11 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	812520	746240	811881	712532
TOTAL SETUP COST	1461.4	1628	1461.4	1461.4
TOTAL UNITS LATENESS	1288	1650	1288	1134
TOTAL INVENTORY CARRYING COST	37200200	37000600	37200200	33400200
THEORETICAL INVENTORY CARRYING COST	8737.58	7249.93	8737.58	8737.58
AVERAGE UTILISATION	0.753628	0.767539	0.742674	0.753628
AVERAGE COORDINATION RANGE	805.975	570.175	810.975	553.28

TABLE B.12 RESULTS (PROBLEM 4)

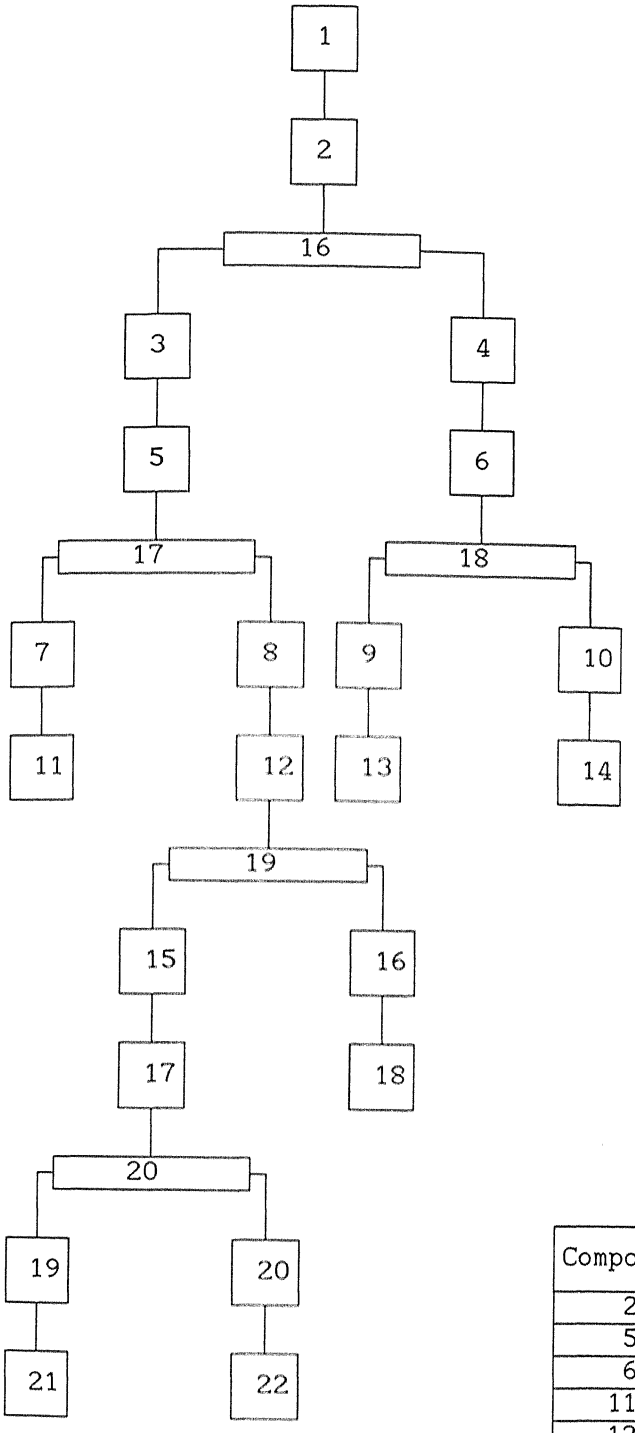


FIGURE B.5 PROBLEM 5

Component No	Routing Sequence
2	1 → 3 → 5
5	1 → 2 → 3
6	3 → 4 → 5
11	1 → 2
12	3 → 4
13	2 → 1
14	4 → 3
17	3 → 4
18	1 → 2
21	5 → 3 → 2
22	4 → 1

TABLE B.13 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
1	13	1,4,6,12,15,16,19,26,31,37,43,48,50
12	4	2,18,26,43
17	5	7,14,21,35,42

TABLE B.14 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	768486	704840	750229	668486
TOTAL SETUP COST	1548	1995	1548	1548
TOTAL UNITS LATENESS	1489	2375	1558	1489
TOTAL INVENTORY CARRYING COST	35000300	52800700	33800300	33000300
THEORETICAL INVENTORY CARRYING COST	8737.58	7249.93	8737.58	8737.58
AVERAGE UTILISATION	0.786595	0.817647	0.768112	0.786595
AVERAGE COORDINATION RANGE	910.5	947.25	934.25	620.8

TABLE B.A5 RESULTS (PROBLEM 5)

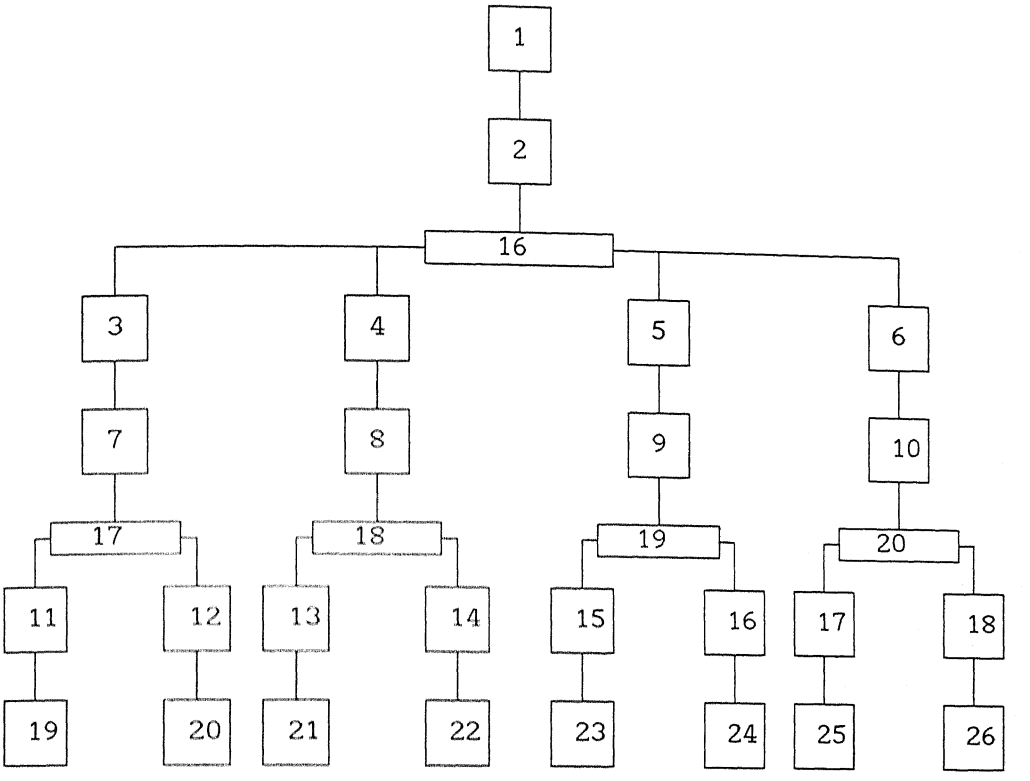


FIGURE B.6 PROBLEM 6

Component No.	Routing Sequence
2	1 → 3 → 5
7	2 → 3 → 4
8	4 → 5 → 1
9	3 → 1 → 2
10	5 → 1 → 2
19	1 → 2
20	3 → 4
21	5 → 1
22	2 → 3
23	4 → 5
24	1 → 2
25	3 → 4
26	2 → 4

TABLE B.16 ROUTING DETAIL OF COMPONENTS

Demand details

Assly No.	No. of demands	Period
1	13	1,4,7,10,12,16,19,26,31,35,41,47,50
4	4	1,21,31,41
6	5	7,22,32,37,42

TABLE B.17 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	954103	958472	978046	972650
TOTAL SETUP COST	2132	2160	2132	2132
TOTAL UNITS LATENESS	1917	1762	1762	1337
TOTAL INVENTORY CARRYING COST	56200400	53800500	58200400	76400400
THEORETICAL INVENTORY CARRYING COST	8804.97	8804.97	8804.97	88049700
AVERAGE UTILISATION	0.849468	0.887055	0.870687	0.811669
AVERAGE COORDINATION RANGE	888	492.5	890.25	488.4

TABLE B.18 RESULTS (PROBLEM 6)

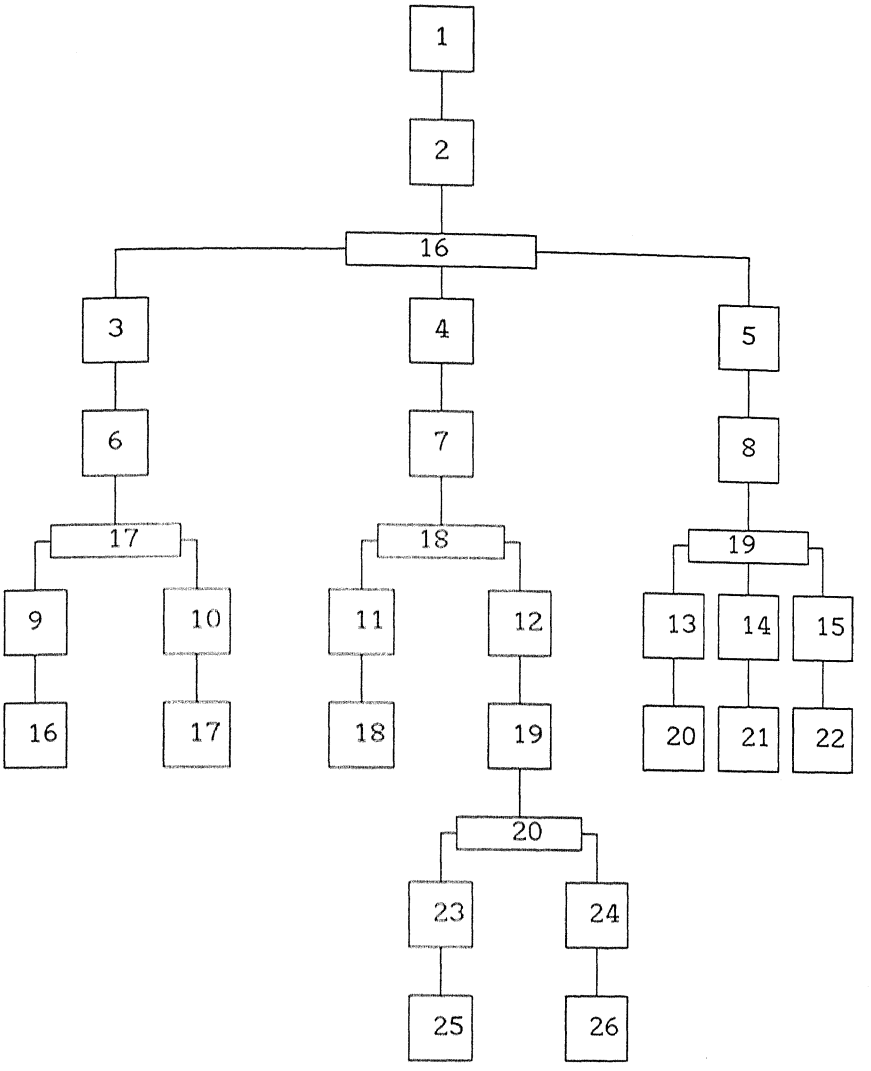


FIGURE B.7 PROBLEM 7

Component No.	Routing Sequence
2	2 → 4 → 5
6	5 → 1
7	2 → 3
8	4 → 1
16	1 → 4
17	2 → 5
18	3 → 5
19	1 → 4
20	4 → 5
21	4 → 3
22	3 → 2
25	2 → 1
26	1 → 3

TABLE B.19 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
1	14	1,7,9,14,16,21,23,27,31,39,41,47,50
3	4	1,21,41,49
5	4	2,22,42,47
15	4	3,11,25,31
23	4	7,34,36,47

TABLE B.20 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	1365700	1327340	1365700	1265000
TOTAL SETUP COST	1932	2006	1932	1932
TOTAL UNITS LATENESS	1392	796	1392	1317
TOTAL INVENTORY CARRYING COST	54800300	74600100	54800300	66000100
THEORETICAL INVENTORY CARRYING COST	8670.92	8536.05	8670.92	8670.92
AVERAGE UTILISATION	0.646617	0.639394	0.646617	0.655988
AVERAGE COORDINATION RANGE	810.75	565.25	810.75	466.4

TABLE B.21 RESULTS (PROBLEM 7)

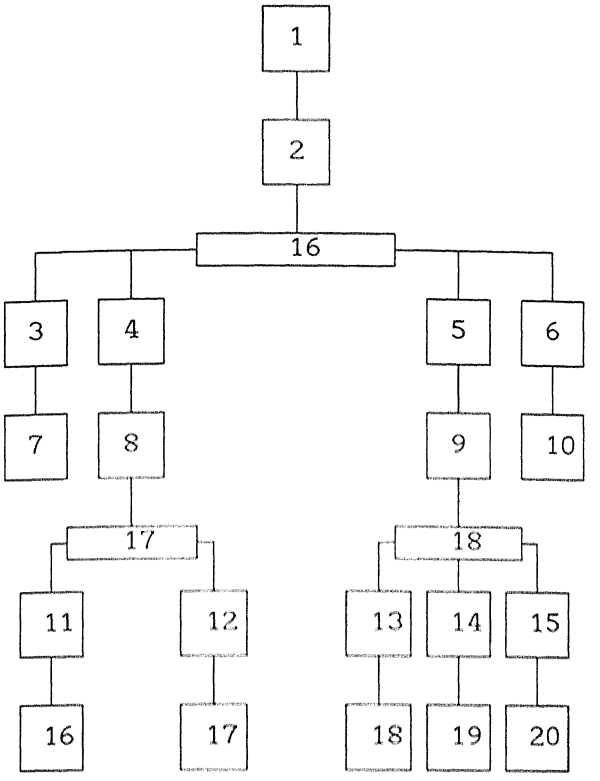


FIGURE B.8 PROBLEM 8

Component No.	Routing Sequence
2	2 → 3 → 4 → 5
7	1 → 2 → 3
8	2 → 3 → 4
9	3 → 4 → 5
10	4 → 5 → 1
16	1 → 2
17	3 → 4
18	5 → 1
19	2 → 3
20	4 → 5

TABLE B.22 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
1	12	1,4,7,11,15,19,24,28,32,38,46,50
4	5	3,7,17,23,43
5	5	2,7,25,39,46

TABLE B.23 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	1049110	1049270	1054680	992817
TOTAL SETUP COST	1804	1804	1804	1804
TOTAL UNITS LATENESS	772	772	728	684
TOTAL INVENTORY CARRYING COST	52600400	52600400	53600400	59000300
THEORETICAL INVENTORY CARRYING COST	8873.66	8873.66	8873.66	8873.66
AVERAGE UTILISATION	0.676471	0.676471	0.680639	0.645474
AVERAGE COORDINATION RANGE	556	548	557.5	161.6

TABLE B.24 RESULTS (PROBLEM 8)

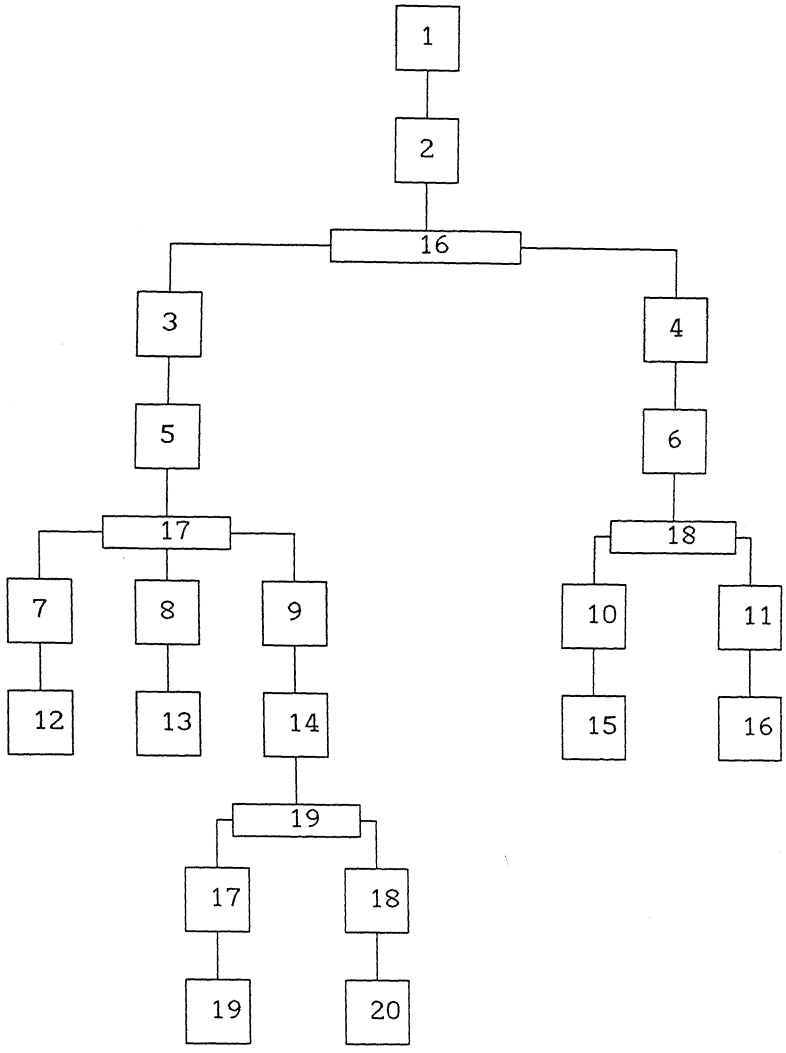


FIGURE B.9 PROBLEM 9

Components	Routing Sequence
2	1 → 4 → 7
5	1 → 2 → 3 → 4
6	5 → 6 → 7 → 4
12	1 → 2 → 3
13	2 → 3 → 4
14	7 → 4 → 2
15	3 → 4 → 5
16	4 → 5 → 6
19	5 → 6 → 7
20	6 → 7 → 1

TABLE B.25 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
7	10	1,4,11,19,22,28,33,39,44,50
9	6	4,8,14,18,21,29

TABLE B_26 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	936183	986948	975974	913242
TOTAL SETUP COST	1548	1683	1548	1548
TOTAL UNITS LATENESS	1260	1118	976	976
TOTAL INVENTORY CARRYING COST	34200100	45800100	37400100	31100000
THEORETICAL INVENTORY CARRYING COST	15393	15393	15393	15393
AVERAGE UTILISATION	0.55773	0.586314	0.587972	0.560045
AVERAGE COORDINATION RANGE	835	571.25	835	274.8

TABLE B.27 RESULTS (PROBLEM 9)

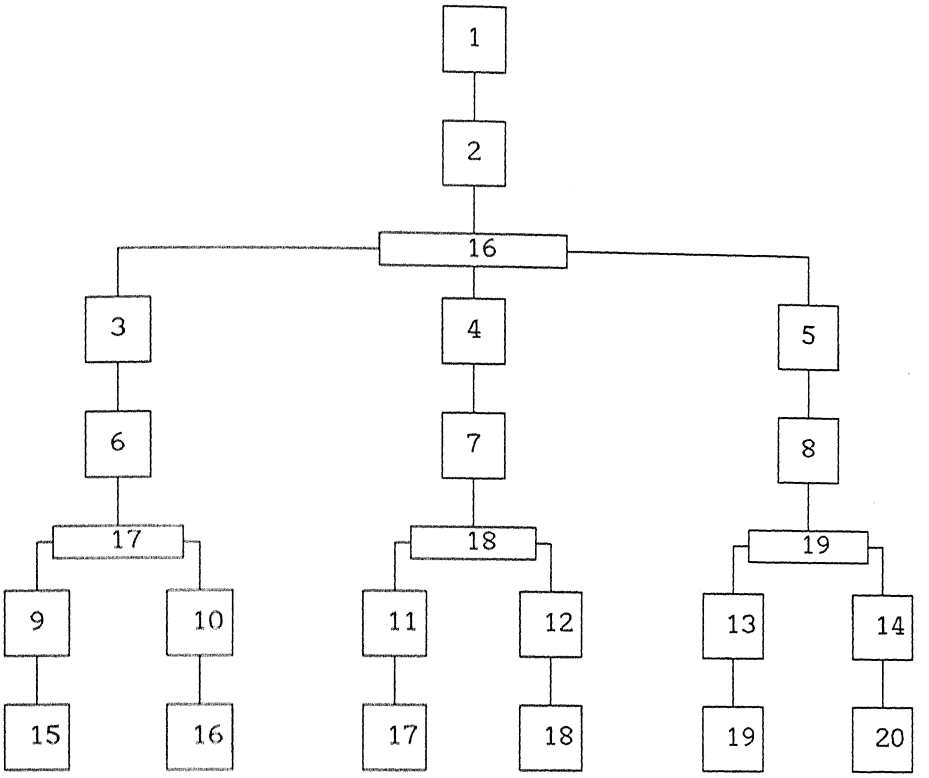


FIGURE B.10 PROBLEM 10

Component No.	Routing Sequence
2	2 → 4 → 6
6	3 → 4 → 5
7	6 → 7 → 1
8	2 → 3 → 1
15	1 → 2
16	3 → 4
17	5 → 6
18	7 → 1
19	2 → 3
20	4 → 5

TABLE B.28 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
1	13	1,3,7,11,16,21,24,27,31,35,41,46,50
9	7	17,27,37,41,43,47,50
12	4	3,11,29,31
14	4	1,21,29,43

TABLE B.29 DEMAND DETAILS

ORIGINAL	H1	H2	H3
705149	748046	692481	671361
1645	1721	1645	1645
756	712	756	701
24400400	27400500	24400400	22600400
11107.7	10528	11107.7	11107.7
0.7443	0.757817	0.7443	0.774761
713.5	371.25	713.5	283.6

TABLE B.30 RESULTS (PROBLEM 10)

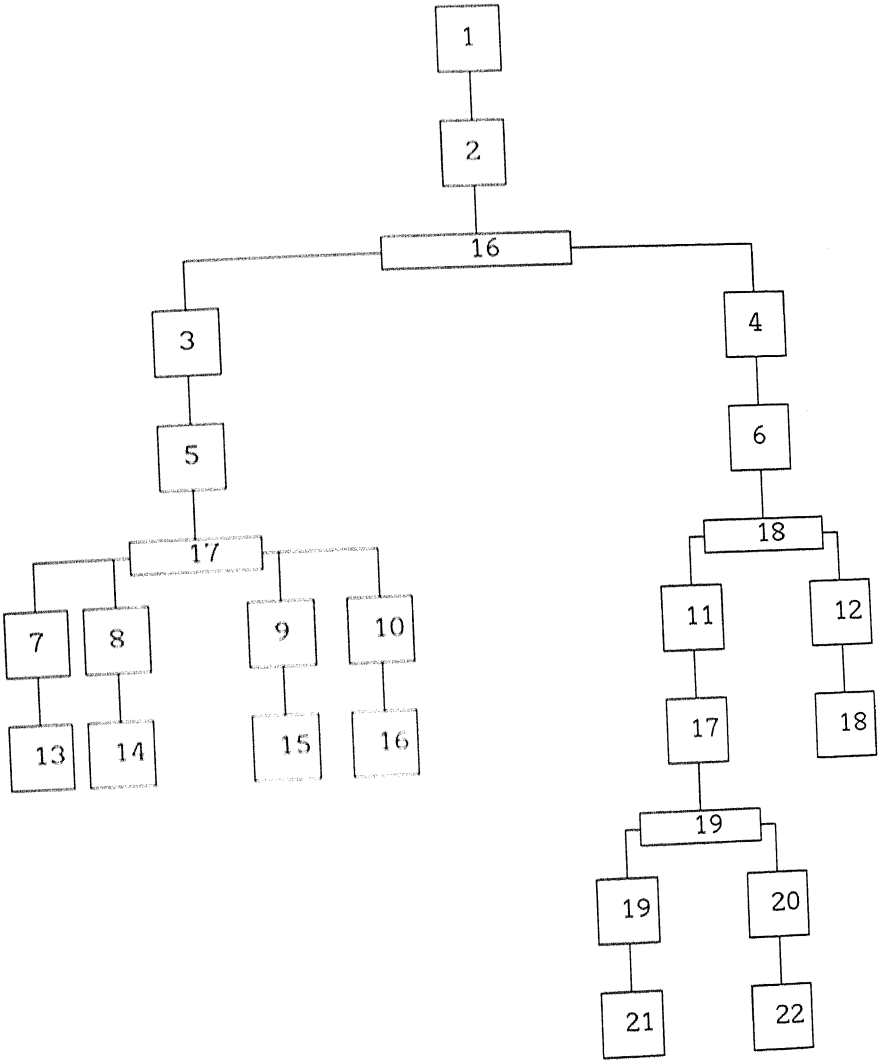


FIGURE B.11 PROBLEM 11

Component No.	Routing Sequence
2	2 → 4 → 5
5	1 → 3 → 4
6	3 → 4 → 2
13	1 → 2
14	3 → 4
15	5 → 1
16	2 → 4
17	4 → 5
18	1 → 2

TABLE B.31 ROUTING DETAIL OF COMPONENTS

Assly No.	No.of demands	Period
1	9	1,4,9,17,21,28,39,44,50
3	4	9,19,29,39
4	6	3,15,22,36,42,46
17	6	8,18,24,30,39,46

TABLE B.32 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	1210380	1210380	1210380	1191560
TOTAL SETUP COST	1538	1538	1538	1538
TOTAL UNITS LATENESS	382	382	382	349
TOTAL INVENTORY CARRYING COST	48800300	48800300	48800300	50600300
THEORETICAL INVENTORY CARRYING COST	8354.61	8354.61	8354.61	8354.61
AVERAGE UTILISATION	0.765354	0.765354	0.765354	0.708108
AVERAGE COORDINATION RANGE	482.5	74.75	482.5	69.6

TABLE B.33 RESULTS (PROBLEM 11)

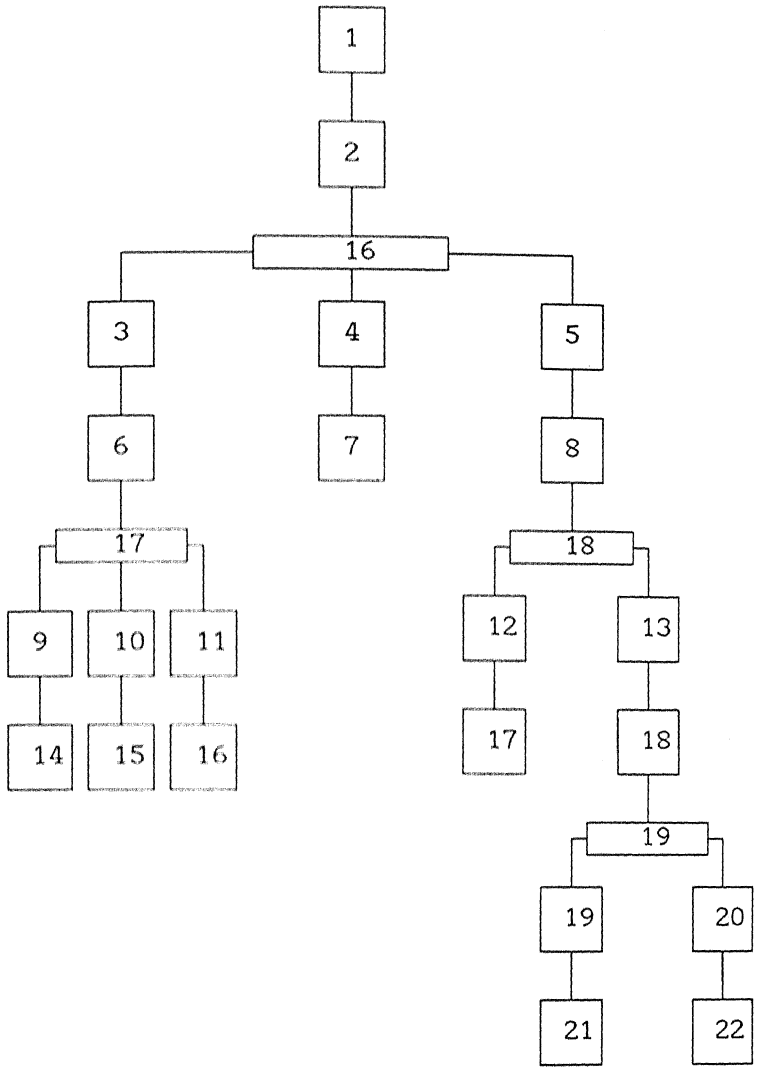


FIGURE B.12 PROBLEM 12

Component No.	Routing Sequence
2	1 → 3 → 5
6	1 → 2
7	3 → 4 → 2 → 1
8	5 → 1
14	2 → 1
15	5 → 2
16	4 → 3
17	3 → 5
18	4 → 1
21	2 → 4
22	1 → 5

TABLE B.34 ROUTING DETAIL OF PARTS

Assly No.	No.of demands	Period
1	10	1,3,7,11,17,19,25,35,45,50
6	5	9,19,27,35,50
8	5	2,4,32,37,45
13	4	4,21,37,46

TABLE B.35 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	871848	871848	871848	774585
TOTAL SETUP COST	1138	1138	1138	1138
TOTAL UNITS LATENESS	1474	1474	1474	1282
TOTAL INVENTORY CARRYING COST	23200400	23200400	23200400	23600300
THEORETICAL INVENTORY CARRYING COST	16233.8	16233.8	16233.8	16233.8
AVERAGE UTILISATION	0.711028	0.711028	0.711028	0.783936
AVERAGE COORDINATION RANGE	694	142.25	694	87.2

TABLE B.36 RESULTS (PROBLEM 12)

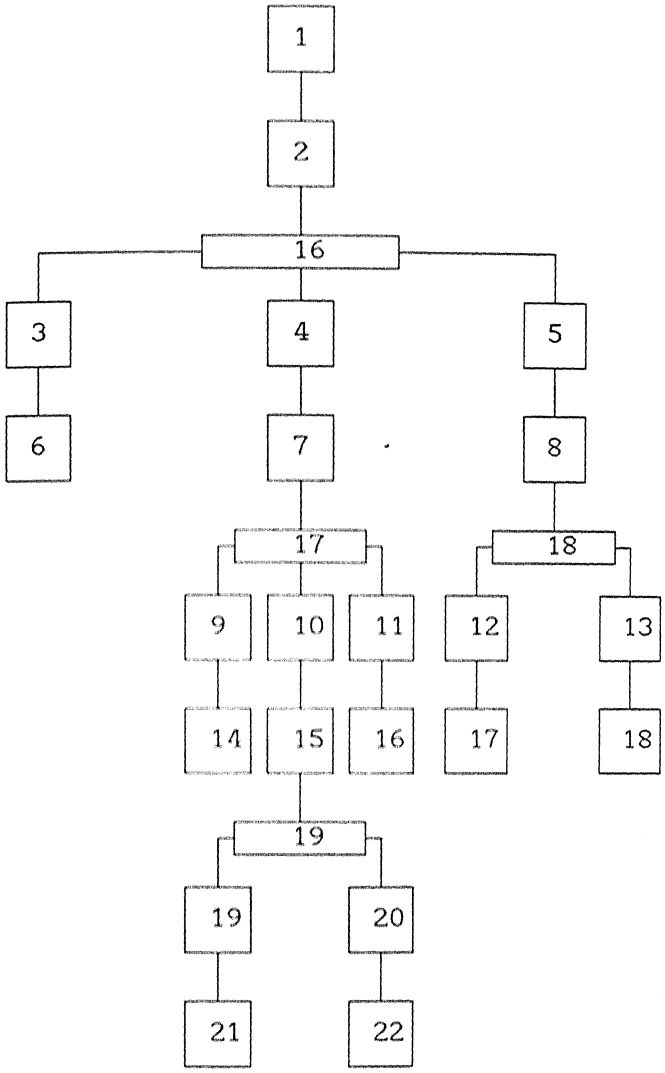


FIGURE B.13 PROBLEM 13

Component No.	Routing Sequence
2	1 → 4 → 5
6	2 → 3
7	1 → 4
8	5 → 4
14	5 → 1
15	3 → 5
16	4 → 2
17	3 → 4
18	1 → 3
21	1 → 4
22	2 → 4

TABLE B.37 ROUTING DETAIL OF COMPONENTS

Demand details

Assly No.	No.of demands	Period
1	10	6,11,17,22,27,31,38,41,46,50
4	6	7,11,19,27,31,41
8	6	4,14,19,27,35,46
10	6	3,13,27,37,41,50

TABLE B.38 DEMAND DEATAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	921864	904073	912091	782381
TOTAL SETUP COST	1392	1412	1392	1392
TOTAL UNITS LATENESS	1937	1462	1651	1418
TOTAL INVENTORY CARRYING COST	211000	242900	92763	8.5643
THEORETICAL INVENTORY CARRYING COST	8372.63	8372.63	8372.63	8372.63
AVERAGE UTILISATION	0.814752	0.793945	0.803269	0.827234
AVERAGE COORDINATION RANGE	1142	279.5	1167.5	136.8

TABLE B.39 RESULTS (PROBLEM 13)

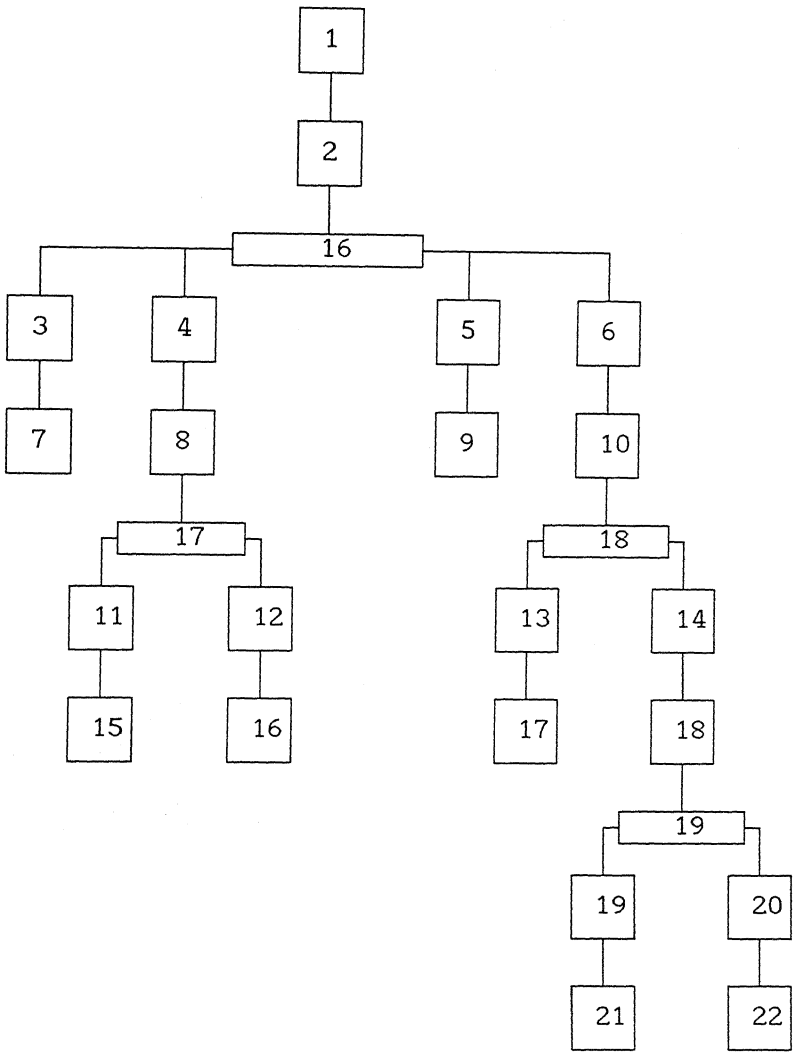


FIGURE B.14 PROBLEM 14

Component No.	Routing Sequence
2	5 → 3 → 1
7	1 → 2 → 3
8	2 → 4
9	2 → 3 → 4
10	3 → 4 →
15	3 → 4
16	4 → 5
17	1 → 2
18	3 → 4
21	4 → 5 → 1
22	5 → 3 → 4

TABLE 40 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
1	10	1,9,21,26,29,31,38,40,46,50
4	7	2,9,21,29,33,41,47
6	5	6,16,26,36,46
14	5	9,14,22,29,38

TABLE B.41 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	938586	938586	943955	804328
TOTAL SETUP COST	1472	1472	1472	1472
TOTAL UNITS LATENESS	1524	1524	1394	1329
TOTAL INVENTORY CARRYING COST	24600400	24600400	24400400	25400400
THEORETICAL INVENTORY CARRYING COST	9548.41	9548.41	9548.41	9548.41
AVERAGE UTILISATION	0.681069	0.681069	0.741236	0.743636
AVERAGE COORDINATION RANGE	1027.5	661.5	1030.25	513.2

TABLE B.42 RESULTS (PROBLEM 14)

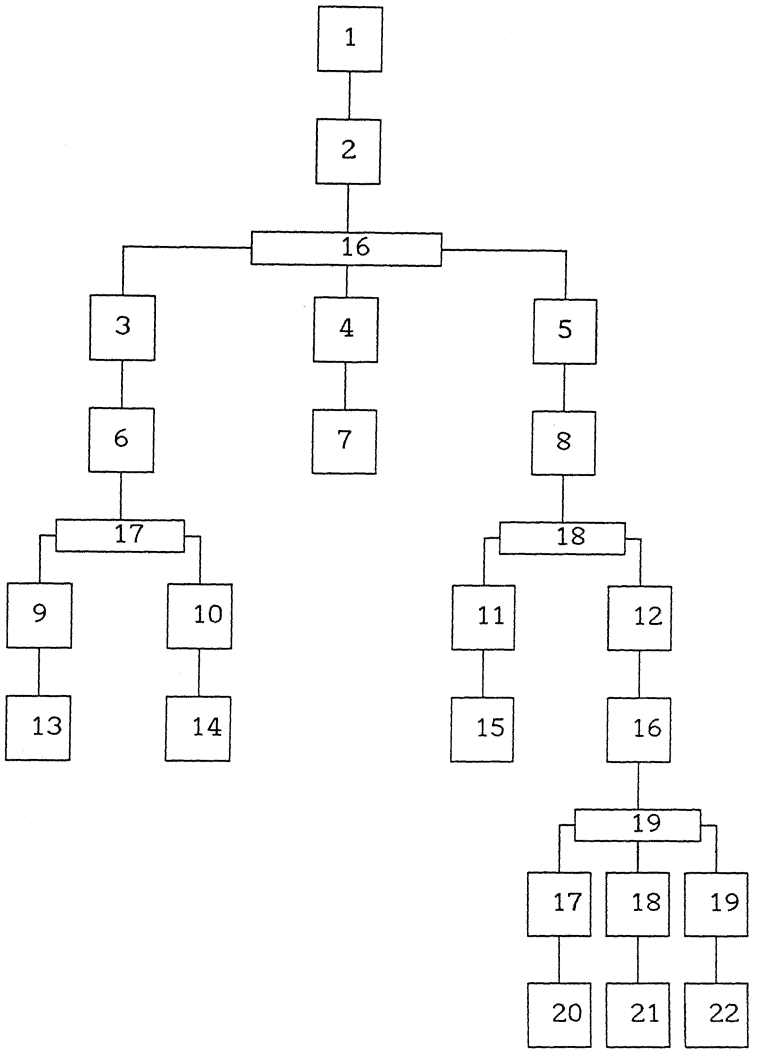


FIGURE B.15 PROBLEM 15

Component No.	Routing Sequence
2	2 → 3 → 5
6	5 → 3 → 2
7	3 → 4 → 5
8	1 → 2 → 3
13	1 → 2 → 3 → 4
14	2 → 3 → 4 → 5
15	5 → 4 → 3 → 2
16	3 → 4 → 1 → 5
20	1 → 2
21	3 → 5
22	4 → 2

TABLE B.43 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
1	9	3,9,15,21,29,32,39,41,48
3	3	9,29,37
5	6	2,11,17,24,31,39
12	5	5,15,25,35,45

TABLE B.44 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	880565	735010	815694	642816
TOTAL SETUP COST	1894	1870	1894	1894
TOTAL UNITS LATENESS	921	2171	1311	270
TOTAL INVENTORY CARRYING COST	311000	337000	237000	213220
THEORETICAL INVENTORY CARRYING COST	8170.46	7146.09	8170.46	8170.46
AVERAGE UTILISATION	0.815569	0.822999	0.726952	0.851916
AVERAGE COORDINATION RANGE	858.5	907	928.5	300.2

TABLE B.45 RESULTS (PROBLEM 15)

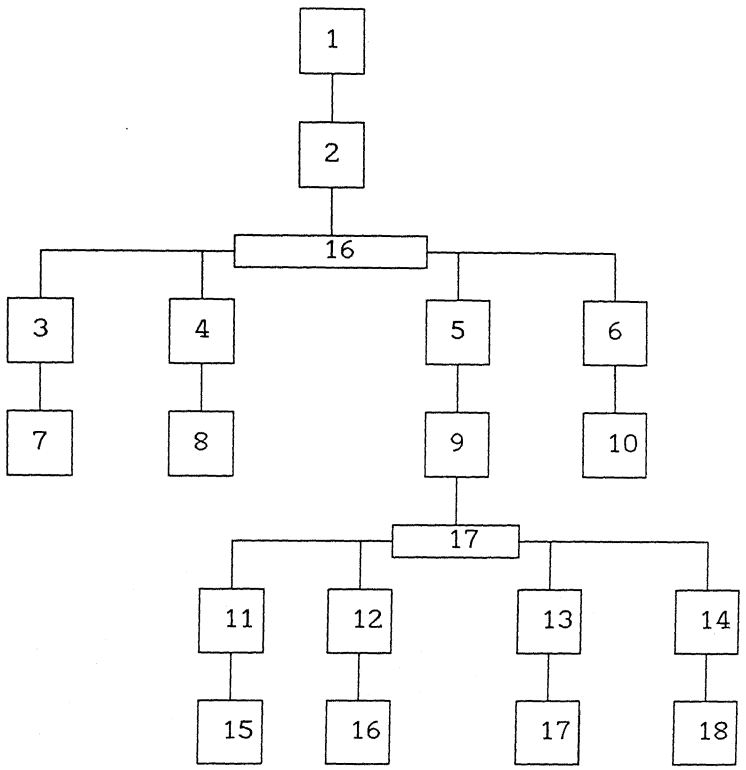


FIGURE B.16 PROBLEM 16

Component No.	Routing Sequence
2	2 → 4 → 6
7	5 → 2 → 3
8	1 → 4 → 2
9	3 → 4 → 5
10	3 → 4 → 1
15	1 → 2
16	3 → 4
17	5 → 6
18	2 → 4

TABLE B.46 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
1	10	1,3,7,17,21,26,29,31,39,47
5	4	9,27,33,47

TABLE B.47 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	583868	644377	587148	560322
TOTAL SETUP COST	895	1288	895	895
TOTAL UNITS LATENESS	582	690	654	548
TOTAL INVENTORY CARRYING COST	21200100	26600200	21200100	21200000
THEORETICAL INVENTORY CARRYING COST	8493.47	10186.9	8493.47	8493.47
AVERAGE UTILISATION	0.748428	0.745731	0.638027	0.802671
AVERAGE COORDINATION RANGE	299.75	166.5	318.25	86.6

TABLE B.48 RESULTS (PROBLEM 16)

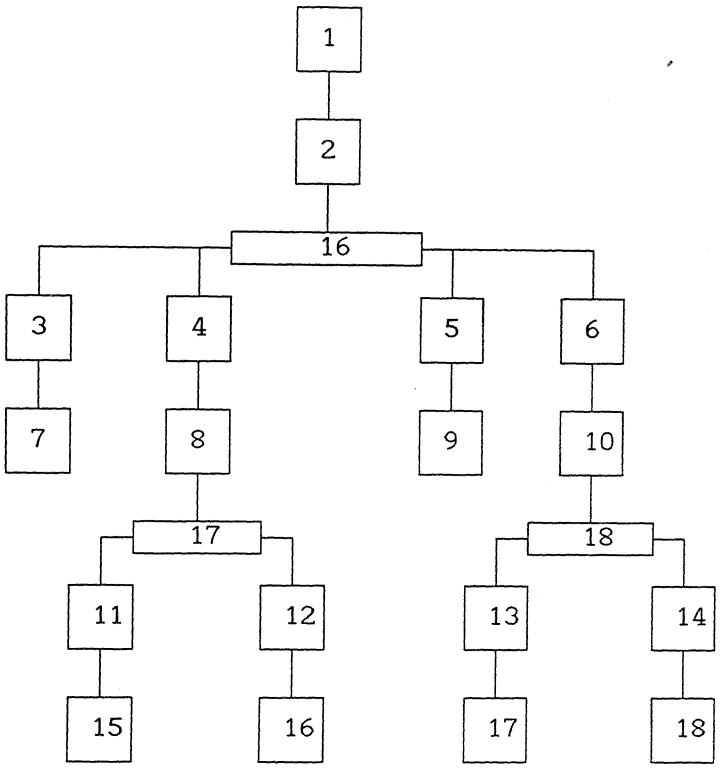


FIGURE B.17 PROBLEM 17

Component No.	Routing Sequence
2	1 → 3 → 6
7	1 → 3 → 5
8	2 → 4 → 6
9	6 → 4 → 2
10	5 → 3 → 1
15	1 → 2
16	3 → 4
17	5 → 6
18	2 → 4 → 6

TABLE B.49 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
1	11	1,3,6,11,15,21,27,31,35,42,50
4	4	4,14,34,44
6	5	6,16,26,36,46

TABLE B.50 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	537844	584176	543596	521826
TOTAL SETUP COST	1372	1781	1372	1372
TOTAL UNITS LATENESS	810	696	753	696
TOTAL INVENTORY CARRYING COST	22200100	23000100	22200100	21400100
THEORETICAL INVENTORY CARRYING COST	4254.23	4254.23	4254.23	4254.23
AVERAGE UTILISATION	0.623706	0.613054	0.680936	0.738272
AVERAGE COORDINATION RANGE	368.75	247.75	388.25	157.2

TABLE B.51 RESULTS (PROBLEM 17)

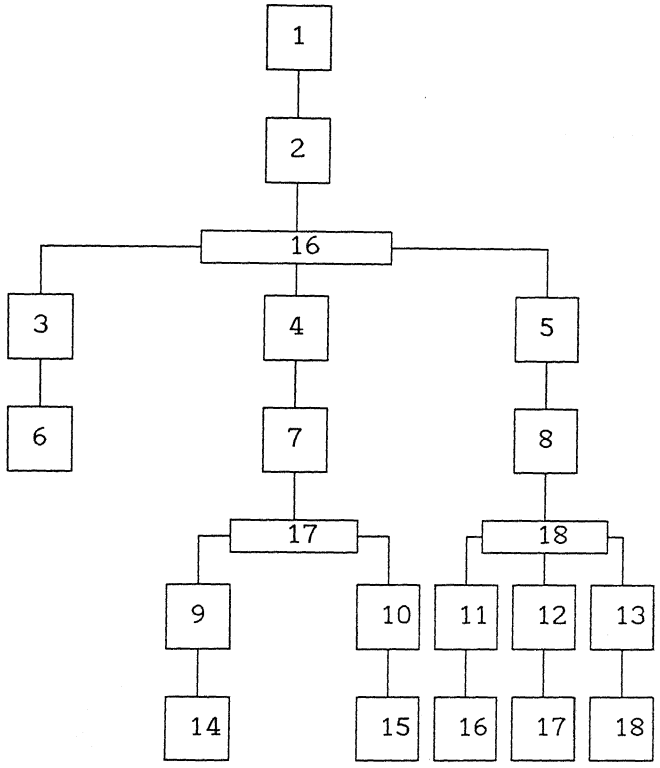


FIGURE B.18 PROBLEM 18

Component No.	Routing Sequence
2	2 → 4 → 5 → 6
6	6 → 5 → 4
7	3 → 5 → 4
8	4 → 5 → 3
14	2 → 1
15	5 → 2
16	4 → 6
17	3 → 6
18	1 → 5

TABLE B.52 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
1	12	1,3,7,11,19,21,28,32,36,42,44,50
4	6	9,14,21,27,35,45
5	5	3,9,16,36,45

TABLE B.53 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	742095	747484	741962	727928
TOTAL SETUP COST	1275	1507	1275	1275
TOTAL UNITS LATENESS	600	634	600	532
TOTAL INVENTORY CARRYING COST	21000400	37000300	21000400	35200200
THEORETICAL INVENTORY CARRYING COST	4816.42	3934.46	4816.42	4816.42
AVERAGE UTILISATION	0.701754	0.623341	0.701754	0.685289
AVERAGE COORDINATION RANGE	432.5	220.25	432.5	142.8

TABLE B.54 RESULTS (PROBLEM 18)

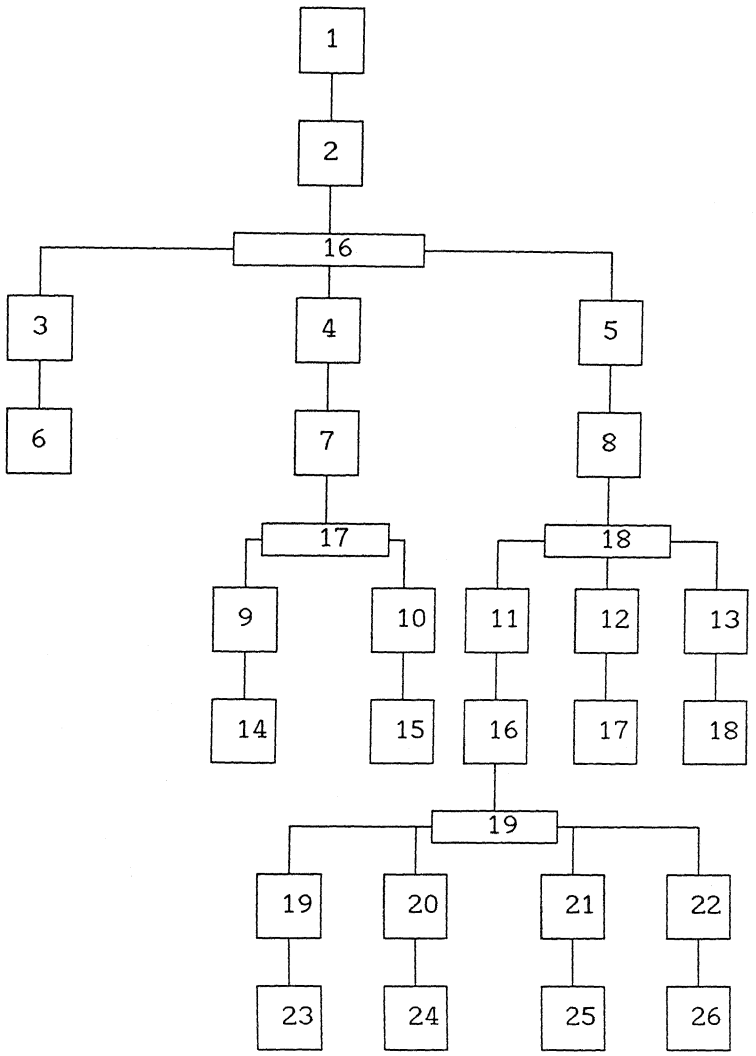


FIGURE B.19 PROBLEM 19

Component No.	Routing Sequence
2	1 → 3 → 5 → 7
6	1 → 2 → 3
7	4 → 5 → 6
8	7 → 2 → 1
14	7 → 3 → 4
15	4 → 5 → 6
16	3 → 4 → 1
17	7 → 6 → 5
18	6 → 4 → 2
23	3 → 1 → 2
24	5 → 6 → 7
25	4 → 6 → 1
26	2 → 5 → 4 → 3

TABLE B.55 ROUTING DETAIL OF COMPONENTS

Assly No.	No. of demands	Period
1	10	1,4,9,17,21,28,33,40,46,50
4	9	3,9,13,19,23,29,33,39,46
5	5	5,15,25,35,45
11	3	21,31,41

TABLE B.56 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	1690930	1758050	1705070	1640580
TOTAL SETUP COST	2534	3918	2534	2534
TOTAL UNITS LATENESS	1283	1632	1283	973
TOTAL INVENTORY CARRYING COST	53600400	65600600	51400400	53200100
THEORETICAL INVENTORY CARRYING COST	13838.3	13838.3	13838.3	13838.3
AVERAGE UTILISATION	0.759847	0.771538	0.743453	0.840568
AVERAGE COORDINATION RANGE	876.75	559.25	889.5	367.2

TABLE B.57 RESULTS (PROBLEM 19)

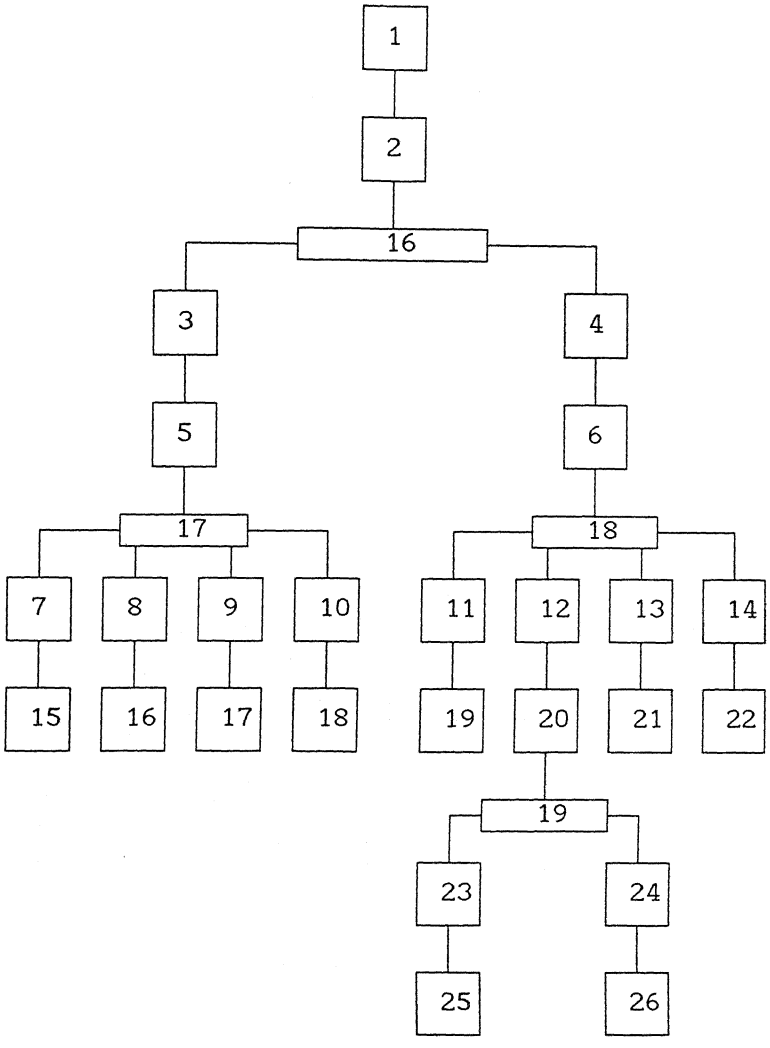


FIGURE B.20 PROBLEM 20

Component No.	Routing Sequence
2	2 → 4 → 6
5	7 → 6 → 5
6	4 → 3 → 2
15	1 → 2
16	3 → 4
17	5 → 6
18	7 → 1
19	3 → 2
20	4 → 5
21	6 → 1
22	2 → 4
25	6 → 3
26	7 → 4

TABLE B.58 ROUTING DETAIL OF COMPONENTS

Assly No	No. of demands	Period
1	12	1,5,9,13,17,21,27,31,35,39,43,47
3	9	6,9,11,17,23,29,33,39,47
4	5	3,13,23,33,43
12	3	19,38,47

TABLE B.59 DEMAND DETAILS

PARAMETER	ORIGINAL	H1	H2	H3
TOTAL AGGREGATE LATENESS	1044140	1120730	1046690	1023130
TOTAL SETUP COST	1833	2217	1833	1833
TOTAL UNITS LATENESS	1214	1159	1214	994
TOTAL INVENTORY CARRYING COST	38200900	44201300	38200900	34600600
THEORETICAL INVENTORY CARRYING COST	22681.6	22117	22681.6	22681.6
AVERAGE UTILISATION	0.728353	0.818681	0.727625	0.855085
AVERAGE COORDINATION RANGE	747.75	523.25	760	317.6

TABLE B.60 RESULTS (PROBLEM 20)

	PERCENTAGE IMPROVEMENT OVER ORIGINAL		
	H1	H2	H3
	0.401055	33.2876	34.70185
	19.01408	7.511737	26.76056
	0	7.23192	12.96758
	−28.1056	0	11.95652
	−59.503	−4.63398	0
	8.08555	8.08555	30.25561
	42.81609	0	5.387931
	0	5.699482	11.39896
	11.26984	22.53968	22.53968
	5.820106	0	7.275132
	0	0	8.638743
	0	0	13.02578
	24.52246	14.7651	26.79401
	0	8.530184	12.79528
	−135.722	−42.3453	70.68404
	−18.5567	−12.3711	5.841924
	14.07407	7.037037	14.07407
	−5.66667	0	11.33333
	−27.2019	0	24.16212
	4.530478	0	18.12191
mean	−7.21111	2.766895	18.43575

TABLE B.61 PERCENTAGE IMPROVEMENT (TOTAL UNITS LATENESS)

H1 Vs H2			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.95	38	1.15128	1.686

H2 Vs H3			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.995	38	3.41075	2.713

H1 Vs H3			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.995	38	2.93549	2.713

TABLE B.62 t TEST DATA (TOTAL UNITS LATENESS)

	PERCENTAGE IMPROVEMENT OVER ORIGINAL		
	H1	H2	H3
	−0.04794	23.87482	26.41635
	−1.7452	−0.60693	3.525107
	−0.01085	−2.52433	−2.61575
	8.157338	0.078644	12.30591
	8.281999	2.37571	13.0126
	−0.45792	−2.50948	−1.94392
	2.808816	0	7.373508
	−0.01525	−0.53093	5.365786
	−5.42255	−4.25034	2.450482
	−6.0834	1.7965	4.791611
	0	0	1.554884
	0	0	11.15596
	1.929894	1.060135	15.13054
	0	−0.57203	14.30428
	16.52973	7.366975	26.9996
	−10.3635	−0.56177	4.032761
	−8.61439	−1.06946	2.978187
	−0.72619	0.017922	1.909055
	−3.96941	−0.83623	2.977651
	−7.33522	−0.24422	2.012182
mean	−0.3542	1.14325	7.686839

TABLE B.63 PERCENTAGE IMPROVEMENT (TOTAL AGGREGATE LATENESS)

H1 Vs H2			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.95	38	0.80593	1.686

H2 Vs H3			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.995	38	2.96734	2.713

H1 Vs H3			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.995	38	4.32772	2.713

TABLE B.64 t TEST DATA (TOTAL AGGREGATE LATENESS)

	PERCENTAGE IMPROVEMENT OVER ORIGINAL		
	H1	H2	H3
	−11.4507	−29.0037	31.35998
	−10.9244	0	1.680672
	0	0	4.977387
	0.536556	0	10.215
	−50.8578	3.428542	2.366843
	4.270254	−3.55869	−31.2713
	−36.1308	0	−20.4375
	0	−1.90113	−10.0744
	−33.918	−9.3567	16.84514
	−12.2953	0	7.376928
	0	0	−3.6885
	0	0	−1.72368
	−15.1185	56.03649	99.99077
	0	0.812995	−4.09829
	−8.36013	23.79421	10.03376
	−25.472	0	0.000472
	−3.60359	0	3.603587
	−76.1885	0	−67.6168
	−22.3883	4.104447	−3.50133
	−15.7075	0	9.424647
mean	−15.8804	2.217821	2.773173

TABLE B.65 PERCENTAGE IMPROVEMENT (TOTAL INVENTORY CARRYING COST)

H1 Vs H2			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.995	38	3.23572	2.713

H2 Vs H3			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.95	38	0.07454	1.686

H1 Vs H3			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.995	38	3.33501	2.713

TABLE B.66 t TEST DATA (TOTAL INVENTORY CARRYING COST)

	PERCENTAGE IMPROVEMENT OVER ORIGINAL		
	H1	H2	H3
	0.587775	39.44973	32.09222
	15.98093	2.823389	8.677738
	0	3.491144	14.53421
	1.845871	−1.4535	0
	3.947648	−2.34975	0
	4.424769	2.497916	−4.44973
	−1.11704	0	1.449235
	0	0.616139	−4.58216
	5.125061	5.422337	0.415075
	1.816069	0	4.09257
	0	0	−7.47968
	0	0	10.25389
	−2.55378	−1.40939	1.532
	0	8.8342	9.186588
	0.91102	−10.8657	4.456643
	−0.36036	−14.7511	7.247591
	−1.70786	9.175798	18.36859
	−11.1739	0	−2.34626
	1.538599	−2.15754	10.62332
	12.40168	−0.09995	17.3998
mean	1.583326	1.961191	6.073582

TABLE B.67 PERCENTAGE IMPROVEMENT (AVERAGE UTILIZATION)

H1 Vs H2			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.95	38	0.46742	1.686

H2 Vs H3			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.9995	38	4.23881	3.57

H1 Vs H3			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.9995	38	5.55442	3.57

TABLE B.68 t TEST DATA (AVERAGE UTILIZATION)

	PERCENTAGE IMPROVEMENT OVER ORIGINAL		
	H1	H2	H3
	0.349502	−0.86316	73.65812
	34.8643	41.75365	61.58664
	39.22948	0	58.78057
	29.25649	−0.62037	31.35271
	−4.03624	−2.60846	31.81768
	44.53829	−0.25338	45
	30.2806	0	42.47302
	1.438849	−0.26978	70.93525
	31.58683	0	67.08982
	47.96776	0	60.25228
	84.50777	0	85.57513
	79.50288	0	87.43516
	75.52539	−2.23292	88.02102
	35.62044	−0.26764	50.05353
	−5.64939	−8.15376	65.03203
	44.45371	−6.17181	71.10926
	32.81356	−5.28814	57.36949
	49.07514	0	66.98266
	36.21329	−1.45423	58.11805
	30.0234	−1.63825	57.52591
mean	35.8781	0.596588	61.50842

TABLE B.69 PERCENTAGE IMPROVEMENT
(AVERAGE COORDINATION RANGE)

H1 Vs H2			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.9995	38	5.789717	3.57

H2 Vs H3			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.9995	38	14.3427	3.57

H1 Vs H3			
k	DEGREES OF FREEDOM	|T| CALCULATED VALUE	t(k) VALUE FROM TABLE
0.9995	38	4.20595	3.57

TABLE B.70 t TEST DATA (AVRAGE COORDINATION RANGE)